UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Danijel Šarić

Android aplikacija za beleženje podatkov pri odvozu komunalnih odpadkov

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Luka Šajn

Ljubljana, 2015

TEMATIKA DELA

Študent naj za diplomsko nalogo izdela Android aplikacijo, ki bi služila komunalnim podjetjem pri odvozu komunalnih odpadkov. Pri odvozu komunalnih odpadkov prejme voznik mesečni seznam strank, na katerega za vsako stranko posebej zapisuje katerega dne je odpeljal določeno vrsto komunalnih odpadkov. Ob velikem številu strank (nekaj sto) se nabere preveč papirjev in tako onemogoča vozniku hitrejše in učinkovitejše delo. Prav tako pa je potrebno vse podatke mesečno ročno vpisovati v podatkovno bazo. Aplikacija bo rešila te težave, saj bo vsak posamezen voznik dobil unikatno uporabniško ime, geslo in seznam strank. Uporabnik bo lahko hitreje našel želeno stranko s pomočjo spustnega menija. Ob izbiri stranke se bo prikazal seznam vrst odpadkov v obliki tabele, v kateri so vnosna polja za količino in težo odpadkov ter potrditveno polje za pranje zabojnikov. Po vnosu količine, teže in označitve potrditvnega polja, bo uporabnik stisnil gumb podpis stranke, kjer se bo stranka podpisala. Po potrebi bo lahko vnesel tudi določene opombe za odvoz. Ko bo zaključil z vpisovanjem odvoza bo moral pritisniti gumb shrani podatke, kjer mu bo vnesene podatke shranilo na lokalno podatkovno bazo. Ob zaključku delovnega dne se bo uporabnik povezal na brezžično omrežje in posodobil podatke v podatkovni bazi na centralnem strežniku podjetja.

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Danijel Šarić,
z vpisno številko 63100412,

sem avtor diplomskega dela z naslovom:

Android aplikacija za beleženje podatkov pri odvozu komunalnih odpadkov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Luka Šajna;
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela;
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 14. 3. 2015 Podpis avtorja:

Zahvaljujem se mentorju, doc. dr. Luku Šajnu ter delavcem iz podjetja Okolje Piran d.o.o. za nasvete in pomoč pri izdelavi diplomskega dela. Le-te ne bi uspel narediti brez podpore svoje družine, ki mi je omogočila študij v Ljubljani. Zato se jim ob tej priložnosti iskreno zahvaljujem.

Zahvalil bi se rad tudi vsem sošolcem in prijateljem, ki so mi v času študija kakorkoli pomagali. Posebej bi se rad zahvalil Nataliji za neizmerno podporo.

Svojim staršem.

KAZALO VSEBINE

POVZETEK

ABSTRACT

1 UVOD 1

1.1 OPIS PROBLEMA 1

1.2 NAMEN APLIKACIJE 3

2 UPORABLJENE TEHNOLOGIJE PRI RAZVOJU 5

2.1 JAVA 5

2.2 MYSQL IN SQL 5

2.3 SQLITE 6

2.4 XML 6

2.5 PHP 7

2.6 ARHITEKTURA REST 7

2.7 OBJEKTNA NOTACIJA JSON 9

3 RAZVOJNA ORODJA 11

3.1 MOBILNI OPERACIJSKI SISTEM ANDROID 11

3.1.1 PLATFORMSKE RAZLIČICE 12

3.1.2 ARHITEKTURA OPERACIJSKEGA SISTEMA 12

3.2 ECLIPSE 13

3.2.1 ANDROID SDK 13

3.2.2 ADT VTIČNIK 14

3.3 DBDESIGNER 14

3.4 PHPMYADMIN 14

4 RAZVOJ APLIKACIJE 15

4.1 ARHITEKTURA APLIKACIJE 15

4.2 DATOTEKA ANDROID MANIFEST 16

4.3 RAZRED ASYNCTASK 16

4.4 PODATKOVNA BAZA MySQL 18

4.5 GRAFIČNI UPORABNIŠKI VMESNIK 20

4.6 PODATKOVNA BAZA SQLITE 22
4.7 AKTIVNOSTI APLIKACIJE 23
4.7.1 AKTIVNOST PrijavaActivity 25
4.7.2 AKTIVNOST VozilaActivity 27
4.7.3 AKTIVNOST OdvozActivity 29
4.7.4 AKTIVNOST PodpisActivity 36
4.8 SINHRONIZACIJA PODATKOVNE BAZE MYSQL Z PODATKOVNO BAZO SQLITE NA NAPRAVI 38
4.9 SINHRONIZACIJA PODATKOVNE BAZE SQLITE Z PODATKOVNO BAZO MYSQL NA STREŽNIKU 40
4.10 TESTIRANJE 44
5 SKLEPNE UGOTOVITVE 45

SEZNAM UPORABLJENIH KRATIC IN SIMBOLOV

kratica	angleško	slovensko
API	Application Programming Interface	Vmesnik za programiranje aplikacij
HTML	HyperText Markup Language	Jezik za označevanje nadbesedila
JSON	JavaScript Object Notation	Objekna notacija JavaScript
XML	Extensible Markup Language	Razširljiv označevalni jezik
PHP	Hypertetxt Preprocessor	Tričrkovni rekurzirni algoritem
SQL	Structured Query Language	Strukturirani povpraševalni jezik
REST	REpresentational State Transfer	Arhitekturni slog
URL	Uniform Resource Locator	Enolični krajevnik vira
URI	Uniform Resource Identifier	Enotni označevalnik vira
OHA	Open Handset Alliance	Poslovno združenje številnih podjetij
OS	Operating System	Operacijski sistem
IDE	Integrated Development Environment	Integrirano razvojno okolje
SDK	Software Development Kit	Programski razvojni paket
ADT	Android Development Tools	Razvojna orodja za Android

POVZETEK

Cilj diplomske naloge je predstavitev in razvoj Android aplikacije za tablične računalnike za beleženje podatkov pri odvozu komunalnih odpadkov. Namenjena je zaposlenim v podjetju Okolje Piran d.o.o., natančneje voznikom tovornjakov komunalnih odpadkov.
V prvem delu diplomskega dela je opisan postopek dosedanjega beleženja podatkov pri odvozu komunalnih odpadkov ter namen aplikacije. V glavnem delu naloge je predstavljen postopek izdelave aplikacije ter njen uporabniški vmesnik. Osrednji del naloge zajema uporabo vseh tehnologij, programskih jezikov in programskih orodij pri razvoju aplikacije. V zadnjem delu diplomske naloge je predstavljen zaključek ter spoznanja, ki so rezultat analize razvoja aplikacije, ter možnosti nadgradnje in izboljšave za aplikacijo.

Ključne besede: Android aplikacija, razvoj aplikacije, tablični računalnik, ravnanje z odpadki

ABSTRACT

The aim of the thesis is to present development of Android application for Tablet PC, which is designed to record data on the removal of municipal waste. The application was developed for employees in the Enterprise Environment, Piran d.o.o. and specially designed for truck drivers of municipal waste.

The first part of the thesis describes technologies and tools used to record data on the removal of municipal waste and the purpose of application. The main part of the thesis is based on the methodology and the idea of application development, followed by a detailed description of user interface. The second part of the thesis describes all technologies, programming languages and software tools related to application development. In conclusion we point out the insights resulted in the application development and present a business plan for future improvements and upgrades of Android application.

Key words: Android application, application development, Tablet PC, waste treatment

1 UVOD

Živimo v času, ko so mobilni telefoni skupek mnogih digitalnih naprav. Tako mobilne naprave niso več le za telefoniranje in pisanje enostavnih sporočil, ampak so veliko več. Zaradi hitrega razvoja mobilnih aparatov in mobilnih tehnologij se povečuje njihova uporabnost iz generacije v generacijo. Razširjenost uporabe mobilnih aplikacij nam omogoča hitrejši in neposrednejši dostop do storitev.

Vse do samega konca študija na Fakulteti za računalništvo in informatiko nisem vedel v katero smer računalništva se bom podal v svoji prihodnosti. Vedel sem le, da si želim dinamično in zanimivo delo. Prav to pa je razvijanje mobilnih aplikacij za operacijski sistem Android. V času študija smo se dokaj dobro spoznali s programskim jezikom Java. Zaradi tega sem se odločil za izdelavo aplikacije za tablične računalnike z operacijskim sistemom Android o katerem nisem vedel veliko, saj za razvoj uporablja programski jezik Java.

Vse več podjetij išče takšne ali drugačne rešitve v mobilnih tehnologijah, bodisi zaradi večjega zaslužka ali pospešitve določenih storitev. Cilj diplomskega dela je ravno pospešitev shranjevanja podatkov pri odvozu komunalnih odpadkov. Diplomsko delo je sestavljeno iz petih poglavij. V prvem poglavju je predstavljena tematika diplomskega dela in namen izdelave le tega. V drugem in tretjem je opisana uporaba tehnologije ter razvojna orodja. V četrtem poglavju je podrobneje opisan postopek razvijanja aplikacije. Sledi peto poglavje, ki je posvečeno sklepnim ugotovitvam.

1.1 OPIS PROBLEMA

Javno podjetje Okolje Piran d.o.o. je gospodarska družba v lasti Občine Piran. Poslanstvo podjetja je skrb za čisto in urejeno okolje. S svojimi dejavnostmi skrbijo za čistost morja, javnih površin ter urejenost parkov. Pri tem zasledujejo najvišje okoljske standarde, ki jih narekuje evropska in slovenska zakonodaja. Podjetje vodi uprava družbe, ki jo sestavljata direktor kot predsednik uprave ter član uprave.

Dejavnosti sektorja so:

- Ravnanje s komunalnimi odpadki in odlaganje ostankov komunalnih odpadkov.
- Javna snaga in čiščenje javnih površin, urejanje javnih poti, površin za pešce in zelenih površin, pešpoti in drugih javnih poti.
- Vzdrževanje, urejanje pokopališč in oddajanje prostorov za grobove v najem ter pogrebne storitve.
- Urejanje ulic, trgov in cest, ki niso razvrščene med magistralne, regionalne in lokalne ceste, ter postavitev in vzdrževanje prometne signalizacije na teh površinah.
- Izobešanje zastav.
- Vzdrževanje in popravila lastnih komunalnih vozil.
- Izdajanje dovoljenj, določanje višin povračil uporabnikov za priključitev na infrastrukturne objekte, izdajanje soglasij in določanje pogojev k dovoljenjem za posege v prostor in okolje, če ti zadevajo javne infarstrukturne objekte in naprave gospodarskih javnih služb.

V diplomskem delu se osredotočimo na ravnanje s komunalnimi odpadki in odlaganje ostankov komunalnih odpadkov, natančneje sektor snaga.
Na začetku vsakega meseca v podjetju vodja enote pripravi obrazce za vnos podatkov o odvozu komunalnih odpadkov. Vsak obrazec je samo za eno stranko kar pomeni, da je potrebno izdelati veliko število obrazcev za vsak mesec.
Vozniki prejmejo obrazce na papirju A4, s katerimi rokujejo skozi celoten mesec. Zaradi velikega števila obrazcev in dolgega časovnega obdobja rokovanja z njimi obstaja nevarnost izgube obrazca.

Vsak obrazec vsebuje tabelo s številom vrstic glede na število dnevov v mesecu. V tabeli se nahajajo stolpci z imeni posod, stolpci za podpis voznika, podpis stranke ter stolpec za pranje zabojnika. Na vsakem obrazcu se nahaja podatek o mesecu, v katerem odvažamo odpadke, ime stranke, prevzemno mesto zabojnikov ter morebitna frekvenca odvažanja odpadkov ali opombe stranke.
S podpisom voznika identificiramo voznika odpeljanih odpadkov in tako točno vemo kateri voznik je opravil določen odvoz. Na ta način se izognemo iskanju krivca pri morebitnih napakah pri odvozu.
Voznik pri odvažanju odpadkov izbere obrazec za ustrezno stranko na katerega vnese količino, težo ali pranje zabojnikov v ustrezen stolpec in vrstico. Po vnosu se mora tudi vsakič podpisati. Podpis stranke je izbiren, vendar zaželjen, če je le to mogoče. Ta postopek voznik ponavlja vsak delovni dan v mesecu.

Ob koncu meseca voznik odda vodji enote izpolnjene obrazce s podatki o odvozih. Vodja enote mora nato podatke iz vsakega obrazca vseh voznikov ročno vnesti v podatkovno zbirko na strežniku, da lahko iste podatke nato uporabijo pri izračunavanju cen storitev.

1.2 NAMEN APLIKACIJE

Namen aplikacije je poenostavitev in pospešitev vnosa podatkov pri odvažanju komunalnih odpadkov, kakor tudi izboljšanje ažurnosti podatkovne zbirke. Aplikacija je namenjena voznikom tovornjakov komunalnih odpadkov. Grafični vmesnik mora biti enostaven in čim bolj pregleden, saj so uporabniki aplikacije iz različnih starostnih skupin.
Vozniki veliko svojega delovnega časa porabijo pri sestopanju iz vozila in obratno.
V dosedanjem načinu beleženja podatkov pri odvažanju odpadkov so veliko časa porabili tudi pri razvrščanju in iskanju ustreznih obrazcev. V ta namen smo razvili aplikacijo, ki vozniku zmanjša čas postanka pri vsakem odvozu. Podjetje lahko tako naredi več odvozov v istem času. Aplikacija pospeši izvajanje odvoza, saj je uporaba tabličnih računalnikov prikladnejša kot delo s papirji.

Za vstop v aplikacijo se mora voznik najprej prijaviti z dodeljenim uporabniškim imenom in geslom. Sledi izbor vozila glede na vrsto in številko registrske tablice. Na sedežu podjetja mora opraviti sinhronizacijo podatkovne zbirke iz strežnika na podatkovno zbirko v napravi, saj potrebujemo brezžično povezavo. Na ta način prenese vse pomembne informacije, ki mu kasneje služijo pri odvozu komunalnih odpadkov. Hitrost izbiranja ustrezne stranke pospešimo z izvlečnim seznamom. Voznik shranjuje podatke na podatkovno zbirko na napravi, saj gre med vožnjo tudi v kraje, kjer ni dostopa do brezžične povezave z internetom.

Zajete podatke na napravi voznik po koncu delovnega dne na sedežu podjetja z brezžično povezavo prenese na strežnik in tako omogoči podjetju dostop do podatkovne zbirke. Podjetje lahko nemudoma uporablja vnesene podatke, saj ni potrebno čakati do konca meseca za prejem vseh podatkov.

2 UPORABLJENE TEHNOLOGIJE PRI RAZVOJU

Pri razvoju aplikacije smo uporabljali različne tehnologije. V nadaljevanju bomo predstavili katere tehnologije so bile uporabljene.

2.1 JAVA

Programski jezik Java je objektno usmerjen visokonivojski programski jezik. Njegove značilnosti so, da je preprost, večniten, prenosen ter varen. Večina sintakse programskega jezika Java izhaja iz programskih jezikov C in C++, vendar je Java uporabniku prijaznejši programski jezik, saj ima prepostejši objektni model in manj nizkonivojskih ukazov. Je eden izmed najbolj priljubljenih programskih jezikov, še posebej za programiranje aplikacij, ki temeljijo na povezavi odjemalec-strežnik.
Leta 1995 ga je razvil Jasmes Gosling iz podjetja Sun Microsystems.
Java je glavni jezik v katerem se pišejo aplikacije za operacijski sistem Android [1].

2.2 MYSQL IN SQL

MySQL je sistem za upravljanje s podatkovnimi bazami, ki ga je razvilo švedsko podjetje MySQL AB. Začetna verzija je bila izdana leta 1995, trenutno pa je v lasti Oracla. Je odprtokodna implementacija relacijske podatkovne baze, ki za delo s podatki uporablja jezik SQL. Po nekaterih ocenah je najbolj razširjen sistem za upravljanje s podatkovnimi bazami. Potrebno je tudi dodati, da je hiter in zmogljiv, saj ne porabi veliko sistemskih virov. Zaradi tega ga lahko uporabljamo tudi na manj zmogljivi strojni opremi. MySQL smo uporabljali za implementacijo podatkovne baze na strežniku [2].

SQL (ang. *Structured Query Language*) oz. strukturirani povpraševalni jezik za delo s podatkovnimi bazami je najbolj razširjen in standardiziran povpraševalni jezik za delo s podatkovnimi zbirkami. SQL je definiran s standardom ANSI/ISO.
SQL standard se razvija od leta 1986 pa vse do danes. Trenutna najnovejša različica standarda je SQL:2003, ki je bil razvit leta 2003 [3].
SQL je sestavljen iz jezika za definicijo podatkov in jezika za manipulacijo s podatki. SQL omogoča vstavljanje, poizvedbo, posodobitev, brisanje, ustvarjanje in spreminjanje sheme ter nadzor dostopa do podatkov v podatkovni shemi. S pomočjo jezika SQL smo lahko upravljali podatke in arhitekturo podatkovne baze MySQL.

2.3 SQLITE

SQLite je relacijska podatkovna baza realizirana v programskem jeziku C. Leta 2000 jo je zasnoval D. Richarh Hipp z namenom, da za izvedbo programa ni potrebna namestitev SUPB-ja ali zahteva po skrbniku podatkovne baze. SQLite ni ločen proces, do katerega odjemalec dostopa preko aplikacije, ampak je vgrajena v aplikacijo za razliko od drugih SUPB-jev. Priljubljena je za lokalno shranjevanje aplikacij, prav tako pa je prilagodljiva mnogim programskim jezikom. SQLite je zelo hitra ob polnjenju podatkov v podatkovno bazo, saj ne potrebuje mehanizmov, dnevnika sprememb, dnevnika transakcij in posnetkov podatkovne baze [4]. Uporabljajo jo mnogi mobilni operacijski sistemi kot so:

- Blackberry.
- Microsoft Windows Phone 8.
- Apple OS.
- Simbian OS.
- Android.

SQLite podatkovno bazo smo uporabljali za shranjevanje podatkov na napravi in prenos oddaljene podatkovne baze na napravo.

2.4 XML

XML je kratica za razširljiv označevalni jezik (ang. Extensible Markup Language), ki opredeljuje sklop pravil za kodiranje dokumentov v formatu, ki je obenem lažje berljiv človeku ter je strojno berljiv [5]. Cilji oblikovanja XML povdarjajo preprostost, splošnost in uporabnost prek interneta. XML je preprost računalniški jezik podoben HTML-ju, ki nam omogoča format za opisovanje strukturiranih podatkov ali arhitektura za prenos podatkov in njihovo izmenjavo med več omrežji. Zasnova XML se osredotoča na dokumentih, ki se pogosto uporabljajo za predstavitev poljubnih podatkovnih struktur, prav tako pa se uporabljajo tudi v spletnih storitvah.
XML spreminja mnogo aspektov računalništva, še posebej na področju komuniciranja aplikacij in strežnikov. Obstaja možnost razširitve, saj si lahko sami ustvarimo imena etiket (ang. TAG). Zelo je uporaben za komunikacije, saj ima zelo preprosto in pregledno zgradbo. XML smo uporabljali za oblikovanje izgleda aplikacije in postavitev elementov.

2.5 PHP

Hypertext Preprocessor, izvirno orodje za osebno spletno stran (ang Personal Home Page Tools), je razširjen odprtokodni programski jezik, ki se uporablja za strežniške uporabe oziroma za razvoj dinamičnih spletnih vsebin [6]. Pogosto je zapisan v povezavi s HTML-jem vendar v nasprotju s HTML stranjo, strežnik PHP skripte ne pošlje neposredno odjemalcu ampak ga razčleni sam pogon PHP. Podoben je običajno strukturiranim programskim jezikom, najbolj jezikoma C in Perl, tako najbolj izkušenim programerjem dovoljuje razvijanje zapletenih uporab brez dolgega učenja. PHP je bil pri razvoju aplikacije vmesni člen med komunikacijo z mobilno napravo in podatkovno bazo na strežniku.

2.6 ARHITEKTURA REST

REST (ang. REpresentational State Transfer) je arhitekturni slog, ki se pogosto uporablja v razvoju spletnih storitev. Njegov namen je definiranje principov, ki opisujejo na kakšen način so viri definirani in naslovljeni, pri čemer se uporabljajo metode protokola HTTP.
Arhitekturni slog REST je pogosto izbran pred SOAP (ang. Simple Object Access Protocol), slogom, ker REST ne porabi toliko pasovne širine in je zato primerenjši za uporabo preko interneta. Odjemalec ima popolno svobodo glede izbire obdelovanja podatkov v sporočilnem sistemu, saj so informacije o virih izpostavljene pri ponudniku storitve, pri tem pa so v sporočilu že informacije, kako naj se procesira. URI ne razkriva postopka kako dostopati do vira, pri čemer je pomemben zgolj naslov. Način procesiranja zahteve na strežniku za uporabnika ni pomemben in omogoča omejen nabor operacij nad viri [7]:

- PUT: Zamenja vir na obstoječi lokaciji ali ustvari nov vir na novi URI lokaciji.
- GET: Brne predstavitve izbranega vira.
- POST: Na obstoječi lokaciji ustvari nov vir.
- DELETE: Izbriše vir na podani lokaciji.

Značilnosti REST spletnih storitev:

- Odjemalec – strežnik (ang. Client-Server):
    - Enoten vmesnik loči odjemalce od strežnikov. To ločevanje poskrbi, da se odjemalcu ni potrebno zavedati podatkovne zbirke, ki jo upravlja strežnik interno, ko zahteva podatke. Zaradi ločevanja je izboljšana prenosljivost odjemalčeve kode, strežniki pa nimajo uporabniškega vmesnika kar pomeni, da so enostavnejši in bolj razširljivi.

- Brezstanjskost:
    - Vsaka zahteva od odjemalca mora vsebovati vse potrebne podatke za razumevanje zahteve. Odjemalec beleži stanje seje in tako ne more izkoristiti vseh shranjenih stanj na strežniku.
- Predpomnenje:
    - Predpomnilnik pomaga izboljšati odgovore iz omrežja in jih označi za prenos v predpomnilnik kot primerne (ang. cacheable) ali kot neprimerne (ang. non-cacheable). Dobro predpomnenje lahko delno ali popolnoma odstrani potrebo po nekaterih povezavah odjemalca s strežnikom. Posledično se še dodatno izboljša zmogljivost in povečljivost.
- Večplastnost:
    - Odjemalec ne more zanesljivo vedeti, ali je neposredno povezan na strežnik ali na posrednika. Posredniški strežnik lahko izboljša razpoložljivost tako da, uravnovesijo obremenitev in omogočijo deljeni predpomnilnik.
- Izvršna koda na zahtevo:
    - Strežnik lahko začasno spremeni ali doda funkcionalnost odjemalcu z izvajanjem kode v obliki skript in appletov. To je edina izbirna omejitev arhitekture REST.
- Enoten vmesnik:
    - Enoten vmesnik je temeljna omejitev pri načrtovanju storitev REST. Enoten vmesnik poenostavi arhitekturo, jasno loči implementacijo in storitev ter posledično omogoči samostojen razvoj vsakega dela sistema. Enotni vmesnik sledi štirim načelom:
        - Določitev sredstev:

          Posamezna sredstva so opredeljena v zahtevah in se prenašajo med odjemalcem in strežnikom. Sredstva so konceptualno ločena od predstavitve. Za lažje razumevanje, strežnik ne odgovori z rezultatom poizvedbe podatkovne zbirke ampak z enim izmed načinov (objekt JSON, dokument HTML ali XML), ki je odjemalcu v prijaznejši obliki.
        - Samo-opisna sporočila:

          Vsako sporočilo vsebuje dovolj informacij o načinu obravnave tega sporočila. Viri so ločeni od svojega prikaza in njihova vsebina je dostopna v različnih formatih.
        - Hipermedij kot upravljalec stanja aplikacije:

          Hipermedij omejuje možnosti odjemalca in odločitve odjemalca vplivajo na stanje aplikacije. Odjemalec mora poznati le začetni URL, da se lahko odloči za primer povezave glede na metapodatke v povezavi.

- Upravljanje s sredstvi glede na odgovor:
  Odjemalec lahko izbriše ali spremeni vir na strežniku, če ima dovoljenje, ko poseduje predstavitev sredstev vključno z metapodatki.

Arhitekturo REST smo uporabljali za komunikacijo s strežnikom in prenos podatkov iz oddaljene podatkovne baze na lokalno podatkovno bazo na napravi.

2.7 OBJEKTNA NOTACIJA JSON

JavaScript Object Notation je preprost format, ki uporablja berljiv tekst za prenos podatkovnih objektov, ki so sestavljeni iz parov atribut-vrednost. Uporablja se predvsem za prenos podatkov med strežnikom in spletno aplikacijo, kot alternativa za XML.

Čeprav prvotno izvira iz skriptnega jezika JavaScript je objektna notacija JSON neodvisna od programskega jezika. Koda za razčlenjevanje in ustvarjanje JSON podatkov je na voljo v številnih programskih jezikih.

Objektna notacija JSON temelji na dveh univerzalnih strukturah, ki jih poznajo vsi moderni programski jeziki in jih lahko uporabljajo tudi za izmenjavo podatkov. Poznamo zbirko parov, ki vsebuje atribut in ime ter urejen seznam vrednosti [8]. Format JSON smo uporabljali pri pošiljanju povratnih informacij iz strežnika do odjemalca.

3 RAZVOJNA ORODJA

Če smo želeli izpolniti zastavljene cilje, smo morali uporabljati različna razvojna orodja, opisana v nadaljevanju. Za razvijanje mobilnih aplikacij Android sta na voljo dva večja razvojna okolja in sicer Android Studio ter Eclipse. Izbrali smo razvojno okolje Eclipse, saj smo se s slednjim tekom študija spoznali. Za kreiranje konceptualnega modela podatkovne baze smo uporabljali DBDesigner za kasnejše urejanje podatkovne baze na strežniku pa smo uporabljali PHPMyAdmin.

3.1 MOBILNI OPERACIJSKI SISTEM ANDROID

Android je odprtokodni programski jezik in operacijski sistem namenjen mobilnim napravam na dotik. Android je zasnovan na operacijskem sistemu Linux. Začetno razvijanje programske opreme podjetja Android Inc. je podprl Google in ga leta 2005 tudi kupil. V namen prizadevanja skupnega razvoja odprtih standardov na področju telefonije in ostalih prenosnih naprav ter razvoja inovacij, je Google leta 2007 ustanovil poslovno združenje Open Handset Alliance (OHA) in začel boj z drugimi operacijskimi sistemi za mobilne naprave kot so Windows Phone in iOS [9].

Proizvajalci mobilnih naprav so uporabljali lastne operacijske sisteme, s čimer so za razvoj aplikacij zahtevali plačljivo razvojno okolje. Prav tako so priložili svoje aplikacije, kar je upočasnilo razvoj mobilne tehnologije. S prihodom Androida je prišla možnost razvijanja aplikacij v brezplačnem, enotnem razvijalnem okolju z enakimi knjižnicami, kar je pripomoglo k hitrejšemu razvoju mobilnih naprav.

Ker je Android brezplačen in odprtokoden, omogoča vsem uporabnikom vpogled v izvorno kodo. S tem omogoča cenejši in lažji razvoj mobilnih aplikacij vsem zunanjim razvijalcem. Zaradi odprtokodnosti se je Android razširil zelo hitro in je danes vodilni operacijski sistem za mobilne naprave ter pokriva kar 80.7% celotnega tržišča [10].

Google je naredil aplikacijo Trgovina Play (ang. Google Play) za prodajanje in distribucijo aplikacij. Ko uporabnik zaključi z izdelovanjem aplikacije lahko prosto objavi aplikacijo na odprtem tržišču in tam tudi nadzira prodajanje svojega produkta. Prek Trgovine Play si uporabniki operacijskega sistema Android mesečno prenesejo več kot 1.5 milijona aplikacij in iger. V mesecu juliju leta 2014 je bilo uporabnikom na voljo kar 1.3 milijona aplikacij [11].

3.1.1 PLATFORMSKE RAZLIČICE

Android je začel svojo pot k uspehu z beta verzijo izdano septembra 2008. Od takrat pa do danes je bilo izdanih veliko popravkov in funkcionalnosti. Google je razvil več različic operacijskega sistema Android in jih poimenoval po slaščicah:

- 1.5. Cupcake.
- 1.6. Donut.
- 2.0. Eclair.
- 2.2. Froyo.
- 2.3. Gingerbread.
- 3.0. Honeycomb.
- 4.0. Ice cream sandwich.
- 4.1./4.2./4.3. Jelly Bean.
- 4.4. Kit Kat.
- 5.0. Lollipop.

3.1.2 ARHITEKTURA OPERACIJSKEGA SISTEMA

Operacijski sistem Android je zgrajen iz petih elementov: aplikacije, aplikacijskega ogrodja, odprtokodne knjižnice, prevajalnika in jedra Linux.
Jedro Linux nam omogoča:

- Interakcijo s strojno opremo na najnižjem nivoju.
- Upravljanje s pomnilnikom.
- Varnost.
- Mrežno komunikacijo.
- Kontrolo procesov, ki so optimizirani za mobilne naprave.

Odprtokonde knjižnice:

- So temeljni del sistema Android.
- Razvijalci jih uporabljajo za dostop do strojnih komponent naprave. Navaden uporabnik ne more dostopati do njih.
- Imamo dve vrsti knjižnic ene so napisane v programskem jeziku C/C++ druge pa v Javi.

Prevajalnik:

- Sestavljata ga Dalvikov navidezni stroj in jedrne knjižnice Java.
- Dalvikov navidezni stroj je optimiziran za izvajanje v okoljih z nizko porabo energije in majhno porabo pomnilnika in procesne moči.
- Uporablja se za prevajanje aplikacijskih kod.

- Omogoča prenos aplikacij na več različnih prenosnih naprav brez ponovnega pisanja izvorne kode.

Aplikacijsko ogrodje:

- Je sestavljeno iz blokov, ki služijo za neposredno sodelovanje aplikacij.
- Vsebuje upravljalce, ki nudijo aplikacijam različne možnosti.
- S pogledom sistema lahko gradimo grafični vmesnik.
- Ponudniki vsebine nam služijo za deljenje podatkov med aplikacijami.

Aplikacije:

- So vrhnji sloj arhitekture
- Napisane so v programskem jeziku Java.
- Shranjene so v paket s končnico apk.
- Zraven operacijskega sistema dobi uporabnik tudi nekaj prednaloženih aplikacij (npr. odjemalec SMS sporočil, spletni brskalnik, aplikacija za upravljanje stikov, aplikacija za telefonijo, koledar, zemljevid, itd.).
- Požene se v svojem procesu, kar omogoča boljšo porabo pomnilnika in neodvisno delovanje z drugimi aplikacijami.

3.2 ECLIPSE

Pri razvoju aplikacije smo uporabljali razvojno okolje Eclipse. To je integrirano razvojno okolje (ang. integrated development enviroment ali IDE), ki vsebuje delovni prostor in vtičnike s katerimi lahko prilagodimo in razširimo razvojno okolje [12]. Eclipse omogoča razvoj aplikacij v mnogih programskih jezikih kot so: Java, C, C++, Phyton, PHP in še mnogi drugi. Razvojno okolje Eclipse je odprtokodno in brezplačno. Lahko ga namestimo tudi na različnih operacijskih sistemih (Windows, MacOS, Linux), prav tako obstajajo različne verzije za 32–bitno in 64–bitno arhitekturo. Omogoča nam lažje pisanje programske kode, enostavnejši zagon in testiranje le te. Programsko kodo vpisujemo v njegov urejevalnik besedil. S pomočjo ukazov lahko zaženemo in testiramo aplikacijo brez prehoda v drug program.

3.2.1 ANDROID SDK

Razvoj Android aplikacij je proces v katerem se razvijejo aplikacije za operacijski sistem Android. Aplikacije so običajno razvite v programskem jeziku Java s pomočjo Androidovega programskega razvojnega paketa (ang. Software Development Kit), ki vključuje celovit nabor orodij kot so: knjižnice, vodiči, razhroščevalnik, emulator, primeri kode in dokumentacije.
Omogoča nam razvoj aplikacij tudi za starejše verzije platform Android v različnih operacijskih sistemih (Windows, MacOs, Linux). Aplikacije so zapakirane v .apk formatu in so shrenjane v

mapi /data/app na operacijskem sistemu Android (zaradi varnosti je mapa dostopna le s korenskim dostopom). Paket .apk vsebuje .dex datoteke, ki so zagonske datoteke Dalvik, mutimedijske datoteke itd [13].

3.2.2 ADT VTIČNIK

ADT vtičnik (ang. Android Development Tools plugin) je dodatek za razvojno okolje Eclipse, ki omogoča uporabnikom, da lahko ustvarimo nove Android projekte, kreiramo uporabniški vmesnik za Android aplikacijo, dodamo pakete, ki temeljijo na Androidovem API-ju, odpravljamo napake aplikacije z uporabo programskega razvojnega paketa Android in izvažamo .apk datoteke za distribucijo naše aplikacije [14].

3.3 DBDESIGNER

Kadar želimo narediti kompleksno podatkovno bazo SQL potem je DBDesigner zelo uporabno orodje. Omogoča nam zgraditi podatkovno bazo v intuitivnem in enostavnem okolju, kjer imamo vizualno predstavitev tabel in njihovih odnosov, ki jih vsebuje naš projekt. Hitro lahko vidimo polje v tabeli ali kako se ena tabela nanaša na drugo. Ko končamo s sestavljanjem konceptualnega modela podatkovne baze lahko izvozimo shemo podatkovne baze kot .sql datoteko oz. se povežemo s strežnikom na katerem želimo ustvariti podatkovno bazo.
Prav tako lahko tudi uvažamo že obstoječe .sql datoteke ali podatkovne baze iz strežnika. Projekt lahko shranimo v izvorni obliki (XML) in s tem shranimo vse informacije (npr. ne moremo shraniti položajev tabel v .sql datoteko). DBDesigner je razširljiv in je zato primeren za delo z mnogimi podatkovnimi zbirkami na strežnikih. Privzeto vsebuje dva vtičnika, enega za PostgreSQL in drugega za MySQL [15].

3.4 PHPMYADMIN

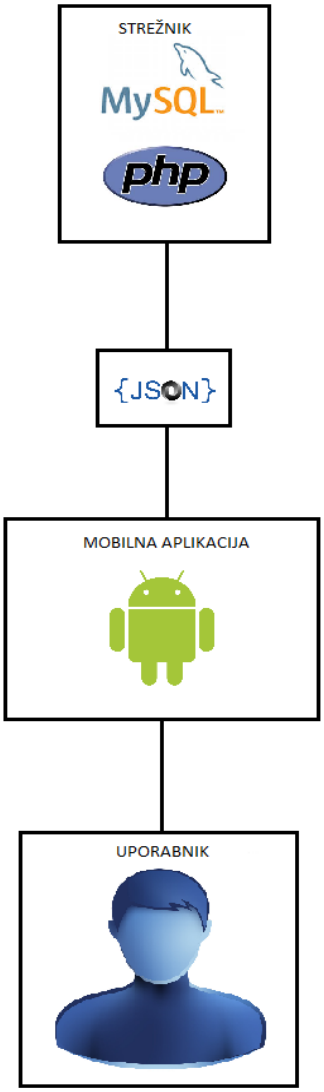
Orodje phpMyAdmin je brezplačno programsko orodje napisano v jeziku PHP in je namenjeno za upravljanje podatkovnih zbirk MySQL prek spleta. Omogoča velik spekter operacij nad MySQL, MariaDB in Drizzle podatkovnimi zbirkami. Pogosto se uporablja operacije (kreiranje, upravljanje in brisanje s podatkovnimi zbirkami, tabelami, stolpci, relacijami, indeksi, uporabniki, dovoljenji, itd.), ki se lahko izvajajo prek uporabniškega vmesnika ali pa neposredno s stavki SQL [16].

4 RAZVOJ APLIKACIJE

V tem poglavju je podrobneje predstavljen rezultat diplomskega dela.

4.1 ARHITEKTURA APLIKACIJE

Aplikacija je sestavljena iz dveh večjih delov: mobilne aplikacije in strežnika (Slika 4.1). Strežnik bo sprejemal in izvajal ukaze, ki jih bo prejel od odjemalca. Zaradi takšne arhitekture je boljša avtonomija podatkovne baze, večja varnost in neodvisnost grafičnega vmesnika. V naslednjih poglavjih so podrobneje predstavljeni vsi deli.



Slika 4.1: Arhitektura sistema.

4.2 DATOTEKA ANDROID MANIFEST

Datoteka *AndroidManifest.xml* je najpomembnejša datoteka Android aplikacije, ki mora biti v korenskem imeniku aplikacije. Vsebuje bistvene informacije o naši aplikaciji na Android operacijskem sistemu. Te informacije mora imeti pred začetkom delovanja katerekoli programske kode naše aplikacije. V datoteki nastavimo verzijo aplikacije, različna dovoljenja, temo, ikono, orientacijo, ime aplikacije ter aktivnosti. Najmanjša podprta SDK verzija je 8, kar pomeni Android OS 2.2, ciljna verzija pa 19, ki stoji za verzijo 4.4 operacijskega sistema Android. Za delovanje naše aplikacije smo potrebovali internet za katerega smo morali določiti dovoljenja za uporabo (Koda 1).

```
<uses-sdk
  android:minSdkVersion="8"
  android:targetSdkVersion="19" />
<uses-permission android:name="android.permission.INTERNET"/>
```

Koda 1: Določitev najmanjše in ciljne SDK verzije ter dovoljenja za uporabo interneta.

Ker je aplikacija namenjena voznikom komunalnih odpatdkov pomeni, da je veliko premikanja samega tabličnega računalnika. V izogib težavam in nevščenostim pri preklapljanju zaslona iz navpičnega v vodoravno postavitev ali obratno smo nastavili vodoravno postavitev aplikacije. To pomeni, da kljub obračanju tabličnega računalnika aplikacija vedno ostane v enaki postavitvi (Koda 2).

```
<activity
  android:name="com.example.diplomska.LoginActivity"
  android:label="@string/app_name"
  android:theme = "@style/Theme.AppCompat.Light"
  android:screenOrientation="landscape">
```

Koda 2: Primer določitve aktivnosti in njegovih lastnosti.

4.3 RAZRED ASYNCTASK

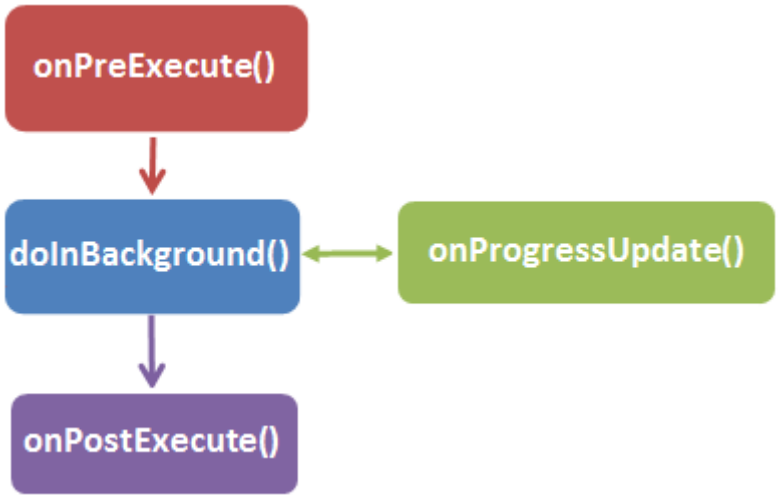
Android upravlja z vnosnimi podatki prek uporabnškega vmesnika na eni niti, ki se imenuje glavna nit (ang. Main thread). Glavna nit ne more opravljati več operacij hkrati, saj opravlja le eno operacijo naenkrat. Če se operacije ne izvajajo sočasno potem se celotna koda naše aplikacije izvaja zaporedno v glavni niti. To pomeni, da se naslednja vrstica kode izvede šele ko se prejšnja konča, kar pa lahko pripelje v dolgotrajno čakanje uporabnika. Če želimo to

preprečiti moramo izvajati vse dolgotrajne operacije asinhrono. Tu pa nam pomaga abstraktni razred *AsyncTask* [17]. Omogoča nam opravljanje dolgotrajnih operacij v ozadju in nato prikaz rezultata na glavni niti brez vpliva na njeno delovanje. V aplikaciji smo ga uporabljali za sinhronizacijo oddaljene podatkovne zbirke na lokalno in obratno. Uporabljali smo ga tudi pri prijavi voznika in izboru vozila, saj smo pri teh operacijah komunicirali s strežnikom.

Nekaj primerov uporabe razreda *AsyncTask:*

- Dostop do virov iz interneta (MP3, JSON, slike).
- Operacije na podatkovni zbirki.
- Klici spletnih storitev.
- Izvajanje zahtevnejših programskih operacij, ki trajajo dalj časa.

Ob začetku izvajanja asinhrone operacije iz glavne niti gre operacija skozi štiri korake (Slika 4.2) s pomočjo metod: *onPreExecute(), doInBackground(), onProgressUpdate()* ter *onPostExecute()*.

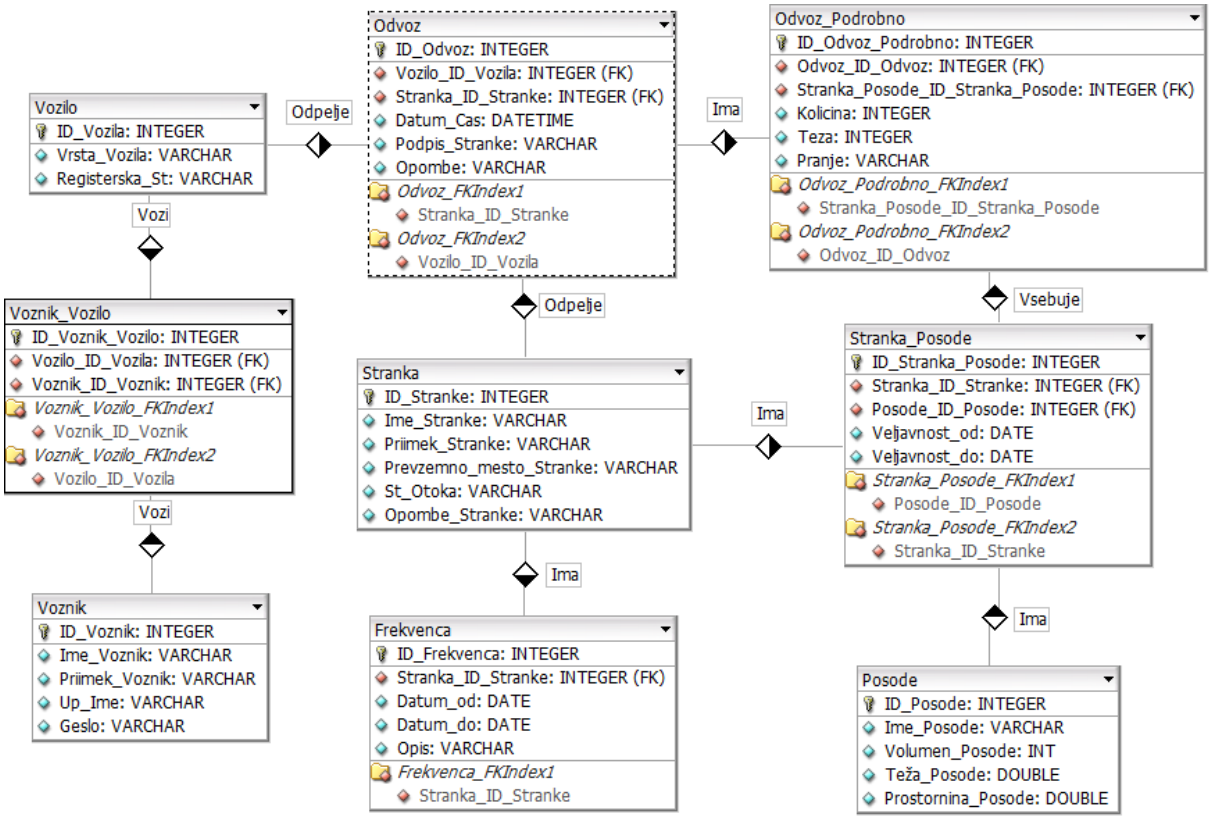


Slika 4.2: Prikaz asinhronega delovanja.

- *onPreExecute()* se izvede pred izvedbo metode *doInBackground()* in je primerna za namestitev nastavitev opravila.
- *doInBackground()* je namenjena za izvajanje kode v ozdaju, ki potrebuje dalj časa za izvedbo.
- *onProgressUpdate()* se lahko uporablja za prikaz kakršnegakoli napredka, ki se izvaja v ozadju in bo prikazan v uporabniškem vmesniku.
- *onPostExecute()* prikaže končne rezultate v uporabniškem vmesniku.

4.4 PODATKOVNA BAZA MySQL

Po sestanku s podjetjem Okolje Piran d.o.o. smo začeli z načrtovanjem podatkovne baze MySQL. Da bi lahko naredili čim boljšo bazo podatkov smo morali z zaposlenimi pregledati vse podatke, ki jih beležijo pri odvozu komunalnih odpadkov in iz tega izluščiti vse pomembne informacije, ki bi bile uporabne pri kreiranju podatkovne baze. Če želimo narediti dobro aplikacijo moramo imeti odlično podatkovno bazo, ki pokriva vse mogoče scenarije. Tako smo v aplikaciji DBDesigner naredili entitetno relacijski model (ERD) podatkovne baze (Slika 4.3).



Slika 4.3:Entitetno relacijski model podatkovne zbirke.

V vsaki entiteti smo implementirali primarni ključ (ID_ in ime entitete), ki se samodejno povečuje in v primeru uporabe služi kot tuji ključ v drugih entitetah.
Entiteto *Voznik* smo smo implementirali, saj smo s tem pospešili postopek odvoza. Namreč voznik se je moral pri vsakem odpeljanem zabojniku odpadkov podpisati. Tako podjetje dobi informacijo o vozniku posameznega zabojnika. Težavo smo rešili z dodeljevanjem unikatnega uporabniškega imena in gesla, ki ju voznik vnese ob prijavi v mobilno aplikacijo. S tem imamo informacijo o vozniku celotnega odvoza in ne le odvoza posameznega zabojnika.

Entiteta V*ozilo* beleži informacijo o *vrsti vozila* in *registrski številki* vozila, ki ga izbere voznik po prijavi v sistem. Entiteta *Voznik_Vozilo* je avtomatsko zgenerirana, saj je povezava med

entitetama *Vozilo* in *Voznik* mnogo proti mnogo (n:m). To pomeni, da lahko en voznik vozi več vozil prav tako pa lahko tudi eno vozilo vozi več različnih voznikov.

V entiteti *Posode* shranjujemo podatke o imenu, volumnu, prostornini in teži posode. Te informacije nam služijo zgolj za prikaz podatkov pri kateri posodi je bila odpeljana določena količina. Kasneje jo lahko uporabimo tudi za obračunavanje stroškov odvoza, saj moramo vedeti iz katerih posod so bile odpeljane smeti.

Entiteta *Stranka* vsebuje naslednje podatke: ime, priimek, prevzemno mesto, številka otoka in opombe. *Prevzemno_mesto* je podatek o lokaciji samega zabojnika. Lahko je naslov najbližjega objekta ali pa je opisno prikazana lokacija zabojnika (npr. pri igrišču). *Številka otoka* je unikatna številka komunalnih otokov, ki ga definira podjetje. *Opombe* stranke je polje za vnos opomb o stranki, kot npr. odvoz samo ob ponedeljkih.

Entiteta *Stranka_Posode* je prav tako avtomatsko zgenerirana, saj ima povezavo med entitetama *Stranka* in *Posode* mnogo proti mngoo (n:m). Takšna relacija pomeni, da ima lahko ena stranka eno ali več posod oz. ena posoda eno ali več strank. Dodali smo dva atributa in sicer *Veljavnost_od* in *Veljavnost_do,* ki nam služita kot časovni razpon veljavnosti določene posode pri določeni stranki. Naprimer lahko se zgodi, da želi stranka prekiniti sodelovanje s podjetjem, kar pomeni, da se pri tej stranki po določenem datumu prekine odvoz odpadkov.

Entiteta *Frekvenca* vsebuje informacije o časovnem razponu odvoza z atributoma *Datum_od* in *Datum_do*. V polje *opis* lahko napišemo tudi frekvenco odvoza zabojnikov, npr. dvakrat na mesec. Frekvenca vsebuje tuji ključ *Stranka_ID_Stranke,* ki se nanaša na entiteto *Stranka.*

Najpomembnejši entiteti sta *Odvoz* in *Odvoz_Podrobno.* V entiteti *Odvoz* beležimo naslednje podatke: *Datum_Cas, Podpis_Stranke* in *Opombe. Datum_Cas* pridobi trenutni čas in datum iz naprave. *Podpis_Stranke* smo morali implementirati, saj se mora vsaka stranka pri odvozu podpisati. V primeru potrebe po iskanju podpisa smo z elektronskim hranjenjem pospešili iskanje, saj nam na ta način ni potrebno iskanje po arhivih datotek. S tujima ključema sta povezani entiteti *Stranka* in *Vozila.* Tako lahko pridobimo informacije o stranki, kateri odvažamo odpadke ter s katerim vozilom jih odvažamo (Slika 4.4).
Entiteta *Odvoz_Podrobno* beleži podatke o količini in teži odpeljanih odpadkov ter podatek o pranju zabojnika. S tujima ključema sta povezani entiteti *Odvoz* in *Stranka_Posode.* Kar nam omogoča informacijo, pri kateri posodi in v katerem odvozu je bila določena količina, teža oz. pranje zabojnika.

Po zaključku izdelovanja ER modela smo ga izvozili v datoteko s končnico .sql, katero smo nato uvozili na strežnik. Za razvijanje in testiranje delovanja aplikacije smo uporabljali lokalni

strežnik, ki ga je poganjal naš osebni računalnik. Po zaključku implementacije smo uporabljali oddaljeni strežnik, ki smo ga nastavili na enemu izmed brezplačnih ponudnikov spletnih strani.

ID_Odvoz_podrobno	Kolicina	Teza	Odvoz_ID_FK	Stranka_Posode_ID_FK	Pranje
1580	3	2000	183	42	true
1581	2	1500	185	37	true
1582	2		186	3	false
1585	1	500	189	1	true
1586	1	700	190	30	false
1587	1	1100	191	1	false
1588	1		192	6	false
1589	1		192	7	false
1590	1		192	8	false
1591	1		192	9	false
1595	3	1085	193	33	true

Slika 4.4: Primer poizvedbe tabele *odvoz_podrobno*.

4.5 GRAFIČNI UPORABNIŠKI VMESNIK

Grafični uporabniški vmesnik (ang. Graphical User Interface oz. GUI) se od leta 1970, ko ga je razvilo podjetje Xerox, ni močno spremenil [18]. Izboljšal se je izgled in dodala kakšna nova vrsta gradnika. Služi kot vmesni člen med uporabnikom in računalnikom oz. programsko opremo. Omogoča njeno uporabo s pomočjo podob neposrednega upravljanja grafičnih slik in elementov z besedilom. Glavna lastnost kakovostnega vmesnika je uporabnost in uporabniška prijaznost. Pri razvijanju grafičnega vmesnika Android imamo na voljo dva načina: proceduralni (tukaj uporabljamo programski jezik Java, kjer podamo niz navodil kako priti do rešitve) in deklarativni (tu je potrebno opisati kaj želimo dobiti kot rezultat in uporabljamo označevalni jezik *XML*).

Kot priporoča Google smo večino projekta naredili na deklarativni način z datotekami *XML*. V primerih, ko smo morali kreirati elemente dinamično, pa smo uporabili tudi proceduralni način. Iz datoteke XML dobimo instanco grafičnega vmesnika s pomočjo metode findViewById() (Koda 3).

```
//vsak grafični element mora vsebovati ID
EditText user = (EditText)findViewById(R.id.editText1);
```

Koda 3: Primer metode *findViewById()*.

Vsakemu elementu grafičnega uporabniškega vmesnika lahko spreminjamo parametre s katerim ga pozicioniramo. Lahko mu nastavimo višino, širino, določimo barvo, velikost prikaza besedila ali nastavimo besedilo (Koda 4).

```
<Button
      android:id="@+id/button1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_below="@+id/editText2"
      android:layout_centerHorizontal="true"
      android:layout_centerVertical="true"
      android:text="@string/Login"
></Button>
```

Koda 4: Primer uporabe parametrov pri gumbu.

V aplikaciji imamo štiri aktivnosti kar pomeni, da moramo imeti štiri različne datoteke xml, ki vsaki aktivnosti določajo izgled. Pri dveh aktivnostih smo uporabljali linearno postavitev, pri dveh pa relativno postavitev elementov. Oba načina postavitev izhajata iz korenskega elementa *View*. Relativno postavitev smo uporabljali pri prijavi in pri izbiri vozila. Linearno pa pri vnosu odvoza in vnosu podpisa stranke. Z relativno postavitvijo lahko določamo, kje bo nastopil kateri element glede na prejšnji ali naslednji element ali relativno na korenski element. Medtem ko pa pri linearni postavitvi razporejamo elemente v vrstici ali stolpcu. To določimo z orientacijo postavitve z lastnostjo *android:orientation*, ki je lahko horizontalna ali vertikalna. Linearna postavitev ima tudi lastnost *weight,* ki nam omogoča izpolnitev korenskega elementa glede na pomembnost elementa. Zelo uporabno je takrat, ko želimo, da se elementi razporedijo v določeni hiearhiji pomembnosti. Če želimo, da sta dva elementa enako velika, nastavimo lastnosti *weight* vrednost 0.5 (*android:layout_weight* = »0.5«).

```
<RelativeLayout
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
```

Koda 5: Primer uporabe relativne razporeditve elementov.

```
<LinearLayout
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:orientation="vertical" >
```

Koda 6: Primer uporabe linearne razporeditve elementov.

Med implementacijo grafičnega uporabniškega vmesnika smo uporabljali različne gradnike. Najpogostejši so bili:

- EditText – vnos besedila,
- Spinner – izvlečni seznam,
- TextView – prikaz besedila,
- Button – gumb,
- CheckBox – potrditveno polje.

4.6 PODATKOVNA BAZA SQLITE

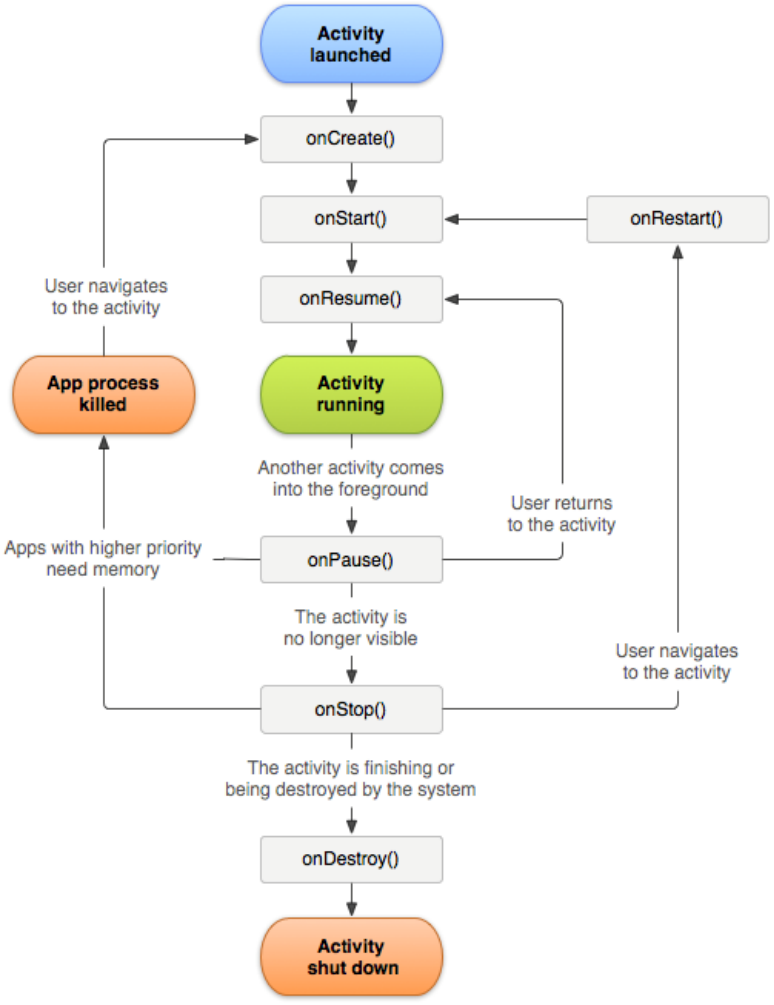
Pri odvozu komunalnih odpadkov se voznik tovornjaka zapelje tudi v odročnejše kraje, kjer ni brezžične povezave. Zaradi tega smo implementirali aplikacijo s tako imenovano »lokalno« podatkovno bazo. Podatkovna baza SQLite je nameščena na vsaki napravi, ki ima operacijski sistem Android. Za potrebe aplikacije smo morali narediti enako podatkovno zbirko kot je na strežniku, saj moramo sinhronizirati podatke na napravo in obratno. Naredili smo nov razred *DBHelper* v katerem smo izdelali (Koda 7), odpirali in posodabljali podatkovno bazo. Razred *DBHelper* vsebuje tudi pomembne metode za opravljanje vseh zahtevnih nalog v povezavi s podatkovno bazo.

```
//SQL stavek za kreiranje tabele stranka
private static final String SQL_CREATE_TABLE_STRANKA = "CREATE TABLE " +
TABLE_STRANKA + "("
      + COLUMN_STRANKA_ID_STRANKA + " INTEGER PRIMARY KEY , "
      + COLUMN_STRANKA_IME_STRANKE + " TEXT NULL, "
      + COLUMN_STRANKA_PRIIMEK_STRANKE + " TEXT NULL, "
      + COLUMN_STRANKA_PREVZEMNO_MESTO_STRANKE + " TEXT NULL, "
      + COLUMN_STRANKA_ST_OTOKA + " TEXT NULL, "
      + COLUMN_STRANKA_OPOMBE_STRANKE + " TEXT NULL "
                  +");";
```

Koda 7: Primer kreiranja podatkovne tabele stranka.

4.7 AKTIVNOSTI APLIKACIJE

Aktivnost (ang. Activity) je komponenta aplikacije, ki omogoča zaslon na katerem lahko uporabnik izvaja operacije (npr. telefoniranje, zajem fotografije, pošiljanje sporočil, itd.). Vsaki aktivnosti je dodeljeno okno na katerem se prikaže uporabniški vmesnik. Običajno okno zasede celotno površino zaslona, lahko pa je tudi manjše in tako rečeno plava (ang. float) nad ostalimi okni. Vsaka aktivnost omogoča zagon nove aktivnosti. Aktivnosti so med seboj neodvisne in samostojne. Aktivnost je del aktivnostnega sklada (ang. activity stack), kjer so na skladu aktivnosti v različnih stanjih (Slika 4.5) [19].

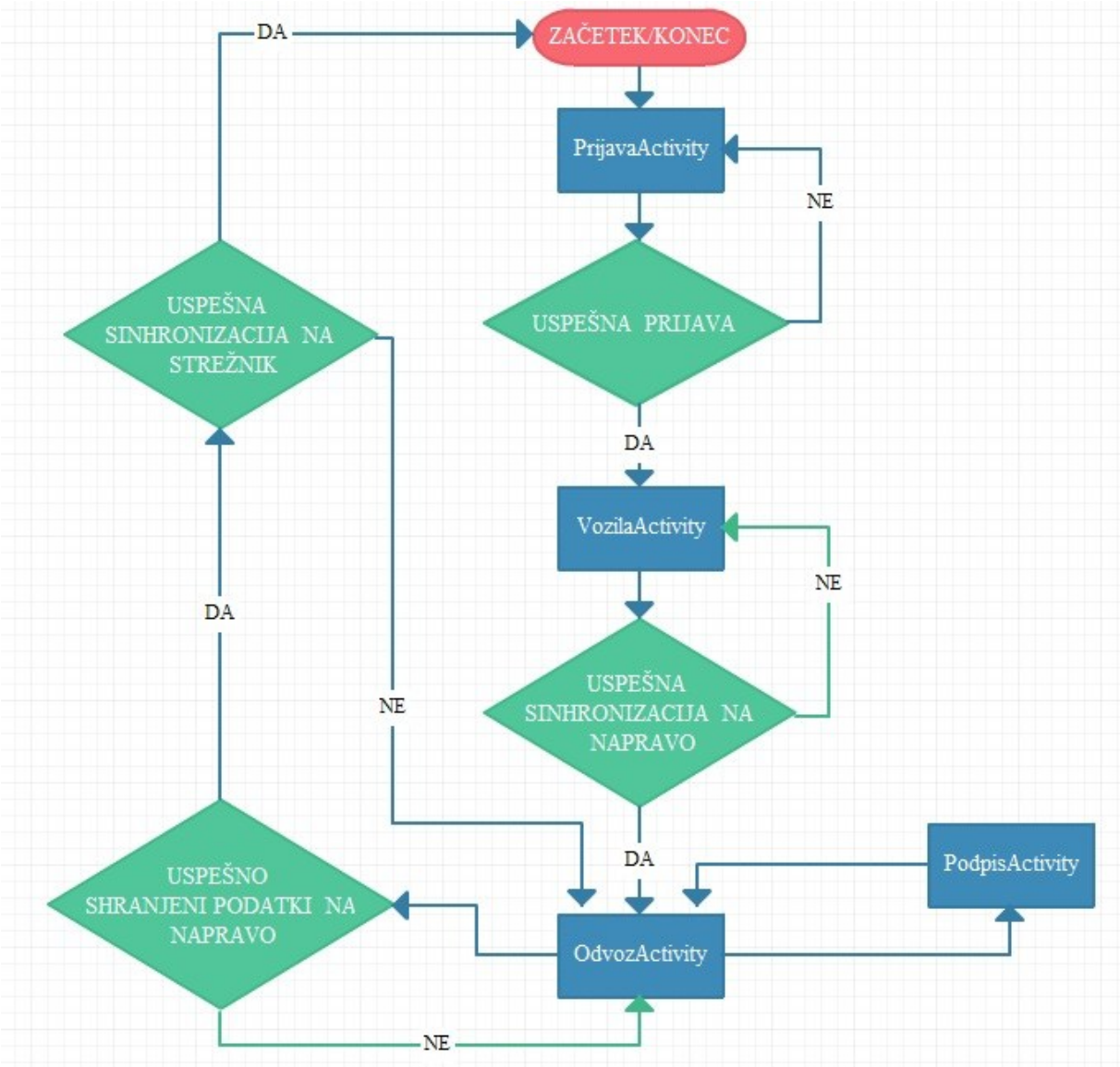


Slika 4.5: Življenski cikel aktivnosti in njihovih stanj.

Naša aplikacija je sestavljena iz mnogih aktivnosti, zato smo morali določiti glavno aktivnost (ang. Main activity), ki se izvede ob zagonu aplikacije.

Aktivnosti naše mobilne aplikacije si sledijo po naslednjem vrstnem redu (Slika 4.6):

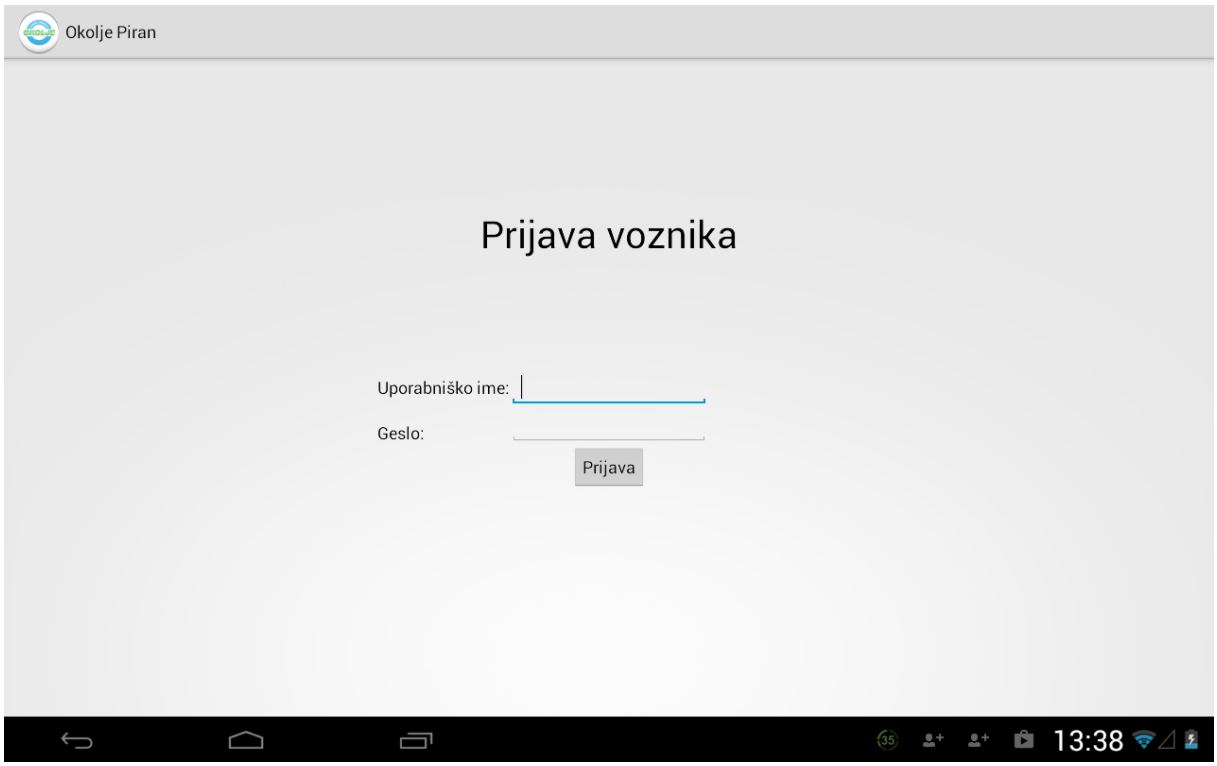
- *PrijavaActivity* (*MainActivity*) omogoča prijavo uporabnika v sistem.
- *VozilaActivity* služi za izbiro vozila.
- *OdvozActivity* prikazuje informacije za odvoz in omogoča shranjevanje podatkov v lokalno podatkovno zbirko.
- *PodpisActivity* je za beleženje podpisa stranke.



Slika 4.6: Diagram aktivnosti.

4.7.1 AKTIVNOST PrijavaActivity

Aktivnost *PrijavaActivity* je uvod v aplikacijo. Ob zagonu aplikacije se nam prikaže kot prva aktivnost. Vsebuje dve vnosni polji, gumb in tri polja za prikaz besedila (Slika 4.7). Voznik za dostop do strežnika potrebuje brezžično povezavo, zato se mora prijaviti na sedežu podjetja.



Slika 4.7: Prijava voznika.

Gumb *Prijava* pritisnemo po vnosu podatkov, ki sproži izvajanje operacij v ozadju za preverjanje pravilnosti vnešenih podatkov.
Po pritisku na gumb v metodi *onClick(View v)* kličemo asinhrono nalogo *AttemptLogin()*.
Metoda *doInBackground(String...args)* v asinhroni nalogi izvede preverjanje vnešenih podatkov. Najprej pošljemo vnešene podatke na strežnik s pomočjo metode *makeHttpRequest(String url, String method, List<NameValuePair> params),* iz razreda *JSONParser,* ki ga pokličemo na začetku dokumenta. V metodo vstavimo tri argumente: URL naslov na katerega se povezujemo (Koda 8), metodo pošiljanja podatkov (POST ali GET) in parametre, ki so vhodni podatki (Koda 9).

```
String LOGIN_URL = "http://danijelsaric.orgfree.com/okoljepiran//login.php";
```

Koda 8: URL naslov na katerega se povezujemo.

URL naslov na katerega se povezujemo se na strežniku navezuje na datoteko *login.php*. Najprej se povežemo na podatkovno bazo MySQL na strežniku in nato preverimo, če se vnešeno uporabniško ime in geslo ujema z uporabniškim imenom in geslom iz podatkovne zbirke.
V objekt JSON (Koda 9), iz katerega na napravi izluščimo podatke, shranimo odgovor o uspešnosti prijave.

```
JSONObject json = jsonParser.makeHttpRequest(LOGIN_URL, "POST", params);
```

Koda 9: Pošiljanje podatkov na strežnik in sprejemanje odgovora v objekt JSON.

V primeru uspešne prijave v splošne nastavitve (ang. *SharedPreferences)* shranimo uporabniško ime, saj ga bomo potrebovali pri shranjevanju odvoza. Razred splošne nastavitve nam omogoča shranjevanje in nalaganje parov ključ-vrednost primitivnih podatkovnih tipov znotraj Android aplikacije. Podatki se samodejno shranijo in obnovijo ob vsaki seji in ob zaključku uporabe aplikacije. Te podatke lahko smatramo kot nekakšne globalne spremenljivke, ki so vedno dostopne. To je za našo aplikacijo idealno, saj imamo več aktivnosti, kjer se lahko podatki pri prehajanju izgubijo.

```
SharedPreferences sp =
PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
Editor edit = sp.edit();
edit.putString("username", username);
edit.commit();
```

Koda 10: Uporaba splošnih nastavitev.

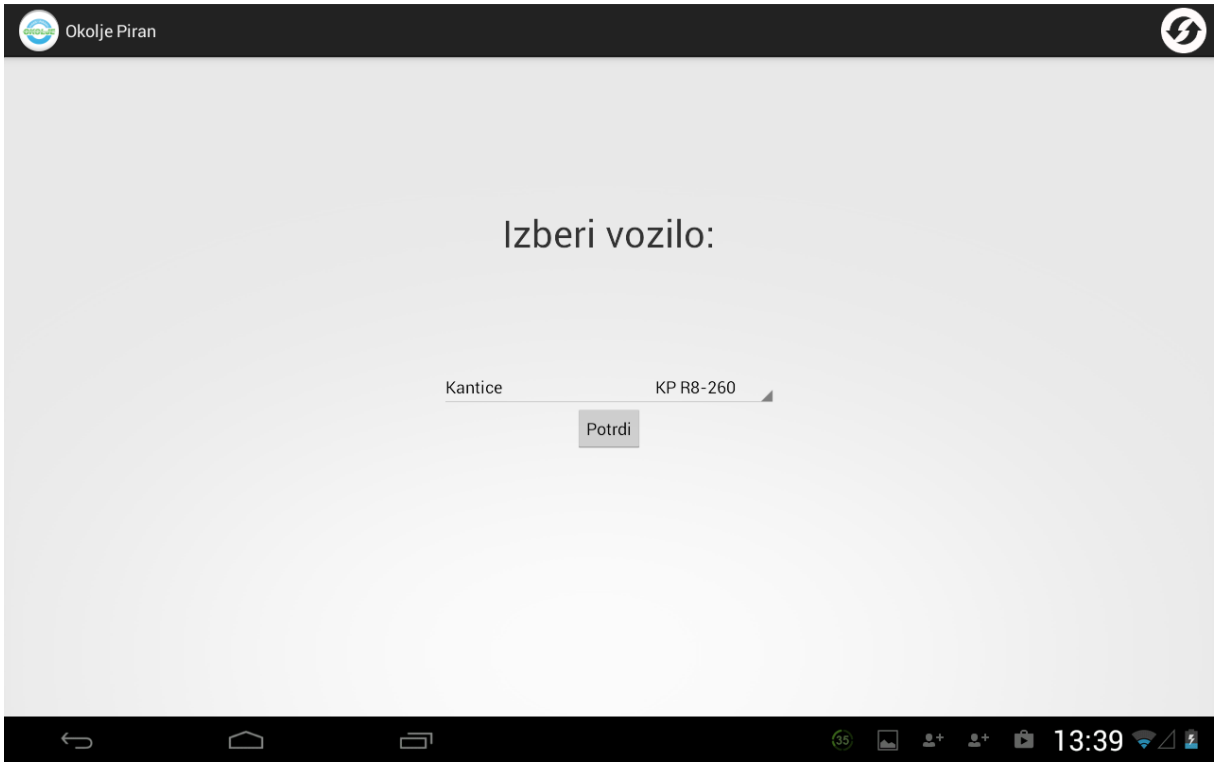
Po shranjevanju podatkov vnesenega uporabniškega imena se izvede namera (ang. Intent) s pomočjo katere preidemo v drugo aktivnost (Koda 11). V vsakem primeru prikažemo informacijo o uspešnosti prijave, preko Androidovega priročnega sistema za sporočila *Toast.*

```
Intent i = new Intent(PrijavaActivity.this, VoziloActivity.class);
finish();
startActivity(i);
```

Koda 11: Način uporabe namere (ang. Intent) za prehod v naslednjo aktivnost.

4.7.2 AKTIVNOST VozilaActivity

Po uspešno opravljeni prijavi v sistem preidemo v aktivnost *VozilaActivity*. V tej aktivnosti opravimo sinhronizacijo oddaljene in lokalne podatkovne zbirke ter izbor vozila. Zaradi sinhronizacije podatkovnih zbirk mora uporabnik to izvesti na sedežu podjetja, saj za komunikacijo s strežnikom potrebujemo brezžično povezavo. Ta aktivnost vsebuje polje za prikaz besedila, izvlečni seznam, gumb za potrditev izbora vozila ter gumb za sinhronizacijo (Slika 4.8).



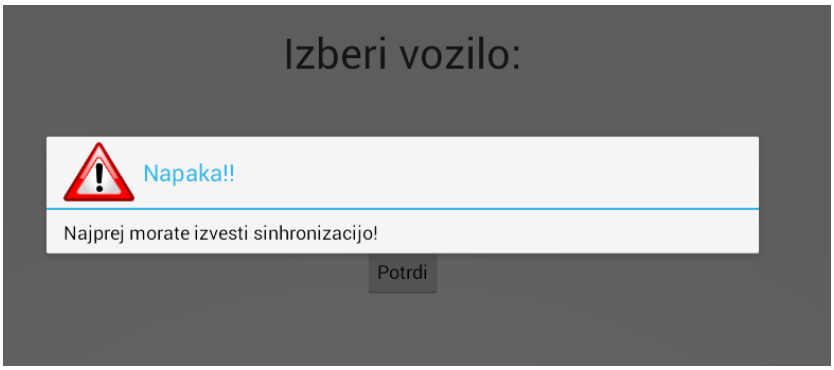
Slika 4.8: Aktivnost *VozilaActivity* ob zagonu.

Ob kreiranju aktivnosti pokličemo razred *DBHelper,* ki izvede kreiranje podatkovne baze *SQLite* na napravi, ter asinhrono nalogo *Vozila()*. Kot v prejšnjem primeru pri prijavi ravno tako izvedemo povezavo na strežnik v metodi *doInBackground(Void... params),* ki vrne *ArrayList<HashMap<String, String>>* kot rezultat izvedbe.
V tej aktivnosti nam ni potrebno pošiljati nikakršnih parametrov, zato to izvedemo z metodo *getJSONFromUrl(final String url)* iz razreda *JSONParser,* ki sprejema kot argument le URL naslov.
Sklicujemo se na datoteko *vozila.php,* ki nam kot odgovor vrne vse podatke o vseh vozilih (identifikacijsko številko vozila, registrsko številko ter vrsto vozila). Te podatke shranimo iz *JSONArraya* v *ArrayList<HashMap<String, String>>* s pomočjo *for* zanke. V metodi *onPostExecute(ArrayList<HashMap<String, String>> arraylist)* nato rezultat iz metode

doInBackground(Void... params) vstavimo v izvlečni seznam. To naredimo s pomočjo enostavnega adapterja, v katerem določimo kateri podatki naj se prikazujejo. Ob kliku na izvlečni seznam se prikažejo podatki pridobljeni iz strežnika (Slika 4.9).

Slika 4.9: Prikaz podatkov v izvlečnem seznamu.

Voznik izbere vozilo, ki se kot uporabniško ime shrani v splošne nastavitve na enak način kot pri prijavi le da tukaj uporabimo metodo *IDVozila(ArrayList<HashMap<String, String>> arrayList),* ki kot rezultat vrne le identifikacijsko številko izbranega vozila (Koda 12).
Če voznik ne izvede sinhronizacije po izboru vozila, mu ne omogočimo prehoda v naslednjo aktivnost s klikom na gum *Potrdi*. Zato aktiviramo opozorilni dialog (ang. Alert Dialog) v katerem izpišemo napako (Slika 4.10).



Slika 4.10: Opozorilni dialog z napisom vrste napake.

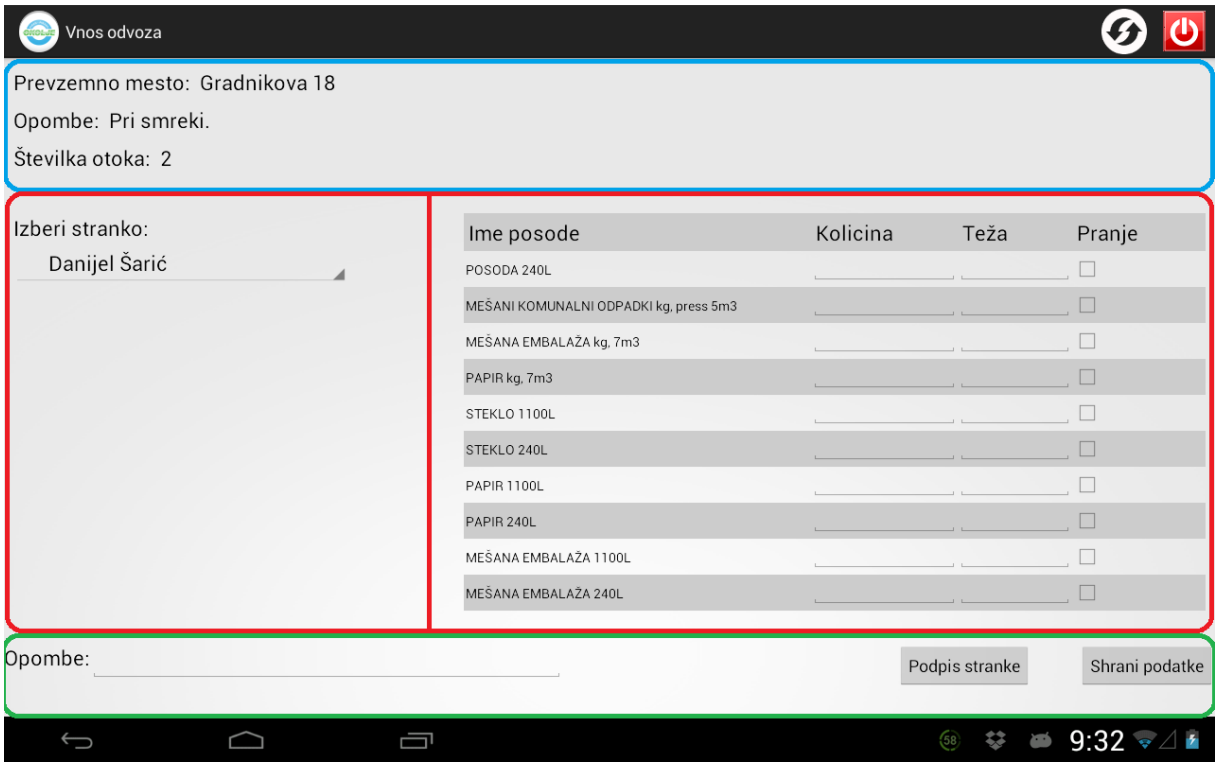
V zgornjem desnem kotu se nahaja ikona za inicializacijo sinhronizacije. Ob kliku na ta gumb se prične sinhronizacija celotne podatkovne zbirke iz strežnika na podatkovno zbirko na napravi. Več o tem v poglavju 5.8 SINHRONIZACIJA PODATKOVNE BAZE MYSQL Z PODATKOVNO BAZO SQLITE NA NAPRAVI. Ob uspešni sinhronizaciji preidemo v novo aktivnost.

```
public String IDVozila( ArrayList<HashMap<String, String>> arrayList){
      String idVozila = "";
      int index = 0;
      for(HashMap<String, String> map : arrayList){
        for(Entry<String,String> entry : map.entrySet()){
            if(sp.getSelectedItemId() == index){
              if(entry.getKey() == TAG_VOZILA_ID){
                  idVozila = entry.getValue();
              }
            }
        }
        index++;
      }
      return idVozila;
}
//klic metode IDVozila nam vrne ID_Vozila izbranega vozila
edit.putString("vozilo", IDVozila(vozilaList));
```

Koda 12: Prikaz metode *IDVozila* in uporaba pri vstavljanju vrednosti v splošne nastavitve.

4.7.3 AKTIVNOST OdvozActivity

Aktivnost *OdvozActivity* je glavna aktivnost mobilne aplikacije. Tukaj se prikazujejo potrebne informacije za voznika in beležijo podatke o odvozu. Grafični vmesnik je razdeljen na tri linearne horizontalne postavitve. Sredinska postavitev je glavna in je razdeljena še na dva vertikalno postavljena dela (Slika 4.11).



Slika 4.11: Glavna aktivnost *OdvozActivity*.

V zgornjem delu aktivnosti (moder okvir) se nam prikazujejo informacije glede na izbor stranke. Dinamično prikazujemo polja: prevzemno mesto, opombe in številko otoka. V primeru, da za določeno polje ni vnosa v podatkovni zbirki, polje umaknemo na proceduralni način kreiranja grafičnega vmesnika (Koda 13).

```
if(opomba == null || opomba.isEmpty()){
      ((TextView)
      findViewById(R.id.opombe_stranke)).setVisibility(View.INVISIBLE);
      ((TextView) findViewById(R.id.prikaz_opombe_stranke))
      .setVisibility(View.INVISIBLE);
}
```

Koda 13: Primer uporabe proceduralnega načina za umik polj.

V srednjem delu (rdeč okvir) imamo izvlečni seznam na levi in tabelo na desni strani linearne postavitve. V tabeli prikazujemo imena posod, vnosni polji za količino in težo ter potrditveno polje za pranje zabojnika.
V desnem zgornjem kotu imamo dva gumba. Prvi z leve aktivira sinhronizacijo lokalne podatkovne zbirke z oddaljeno drugi pa odjavo iz sistema.
Podatke napolnimo v izvlečni seznam iz lokalne podatkovne zbirke s pomočjo metode *getStrankaIDAndStrankaName()*, ki nam vrne identifikacijsko številko in imena vseh strank (Koda 14).

```
public ArrayList<HashMap<String, String>> getStrankaIDAndStrankaName() {
      ArrayList<HashMap<String, String>> strankaIDAndStrankaNameList;
      strankaIDAndStrankaNameList = new ArrayList<HashMap<String, String>>();
      String selectQuery = "SELECT ID_Stranke, Ime_Stranke FROM stranka";
      SQLiteDatabase database = this.getWritableDatabase();
      Cursor cursor = database.rawQuery(selectQuery, null);
      if (cursor.moveToFirst()) {
         do {
            HashMap<String, String> map = new HashMap<String, String>();
            map.put("ID_Stranke", cursor.getString(0));
            map.put("Ime_Stranke", cursor.getString(1));
            strankaIDAndStrankaNameList.add(map);
         } while (cursor.moveToNext());
      }
      database.close();
      return strankaIDAndStrankaNameList;
}
//uporaba v aktivnosti OdvozActivity
SimpleAdapter sAdap = new SimpleAdapter(OdvozActivity.this,
controller.getStrankaIDAndStrankaName(), R.layout.activity_show_stranke,
new String[] {TAG_STRANKA_ID, TAG_IME_STRANKE}, new int[] {R.id.ColIDStranke,
R.id.ColImeStranke});
sp.setAdapter(sAdap);
```

Koda 14: Metoda *getStrankaIDAndStrankaName()* iz razreda ter prikaz njene uporabe.

Po polnjenju izvlečnega seznama moramo prikazati ustrezne podatke. Ob kliku na določeno stranko se prikažejo posode, ki so le pri tej stranki. Zato smo implementirali metodo *getPosode(String id_stranke)* v razredu *DBHelper,* ki s pomočjo argumenta o izbrani stranki iz izvlečnega seznama in vgnezdenih stavkov *SQL* vrne pravilen rezultat (Koda 15).

```
String selectQuery =
"select S.ID_Stranke, S.Ime_Stranke, S.Prevzemno_Mesto_Stranke, S.St_Otoka,
S.Opombe_stranke, SPD.Ime_Posode, SPD.Volumen_Posode, SPD.Teza_Posode,
SPD.Prostornina_Posode, SPD.ID_Stranka_Posode " +
      "from Stranka S " +
      "left join ( " +
            "Select SP.ID_Stranka_Posode, SP.Stranka_ID_FK, SP.Posode_ID_FK,
            P.Ime_Posode,P.Volumen_Posode, P.Teza_Posode,
            P.Prostornina_Posode " +
            "from stranka_posode SP " +
            "join Posode P on SP.Posode_ID_FK=P.ID_Posode
       ) " +
"SPD on SPD.Stranka_ID_FK = S.ID_Stranke " +
"where S.ID_Stranke = ? " +
"order by S.ID_Stranke ";
SQLiteDatabase database = this.getWritableDatabase();
Cursor cursor = database.rawQuery(selectQuery, new String[] { id_stranke });
```

Koda 15: Vgnezdeni stavki SQL v metodi *getPosode(String id_stranke)* iz razreda *DBHelper.*

Pridobljene vrednosti nato uporabimo v metodi *metodaTabela(ArrayList<HashMap<String, String>> arrayList)*. Iz vrnjenega rezultata pridobimo imena posod, ki ustrezajo izbrani stranki in jih shranimo v tabelo nizov (Koda 16). Tabelo za prikazovanje imen posod, vnosnih polj in potrditvenega polja naredimo dinamično, saj se njena velikost spreminja glede na izbrano stranko. Izvajamo proceduralni način kreiranja grafičnega vmesnika s pomočjo programskega jezika Java. V *for* zanki vstavljamo vrstice z elementi toliko časa kolikor je vnosov v seznamu razpršenih tabel.

```
for(HashMap<String, String> map : arrayList){
      Set<String> keys = map.keySet();
      for(String key : keys){
            if(key == TAG_IME_POSODE){
                  arrayImenPosod[count]= map.get(TAG_IME_POSODE);
            }
      }
```

Koda 16: Pridobivanje in shranjevanje imen posod v tabelo nizov.

Ker želimo, da uporabnik v polja za vnos količine in teže vnaša številke, smo določili način vnosa podatkov. Tako se ob kliku na vnosno polje količine ali teže prikaže številčna tipkovnica, ki vsebuje gumb za skok v naslednje vnosno polje (Koda 17).

```
labelTeza.setRawInputType(Configuration.KEYBOARD_12KEY);
labelTeza.setImeOptions(EditorInfo.IME_ACTION_NEXT);
```

Koda 17: Nastavitev odpiranja številčne tipkovnice ob kliku na vnosno polje. Nastavitev funkcije gumba *Naprej*.

Elemente vstavljamo v vrstice, vrstice nato vstavimo v glavno tabelo s pomočjo metode *addView()* (Koda 18). Vsak element se shranjuje v seznam tipa elementa (npr. element količina se shranjuje v seznam vnosnih polj količin).

```
tr.addView(labelPranje);
t1.addView(tr, new
LayoutParams(android.view.ViewGroup.LayoutParams.MATCH_PARENT,
android.view.ViewGroup.LayoutParams.MATCH_PARENT));
```

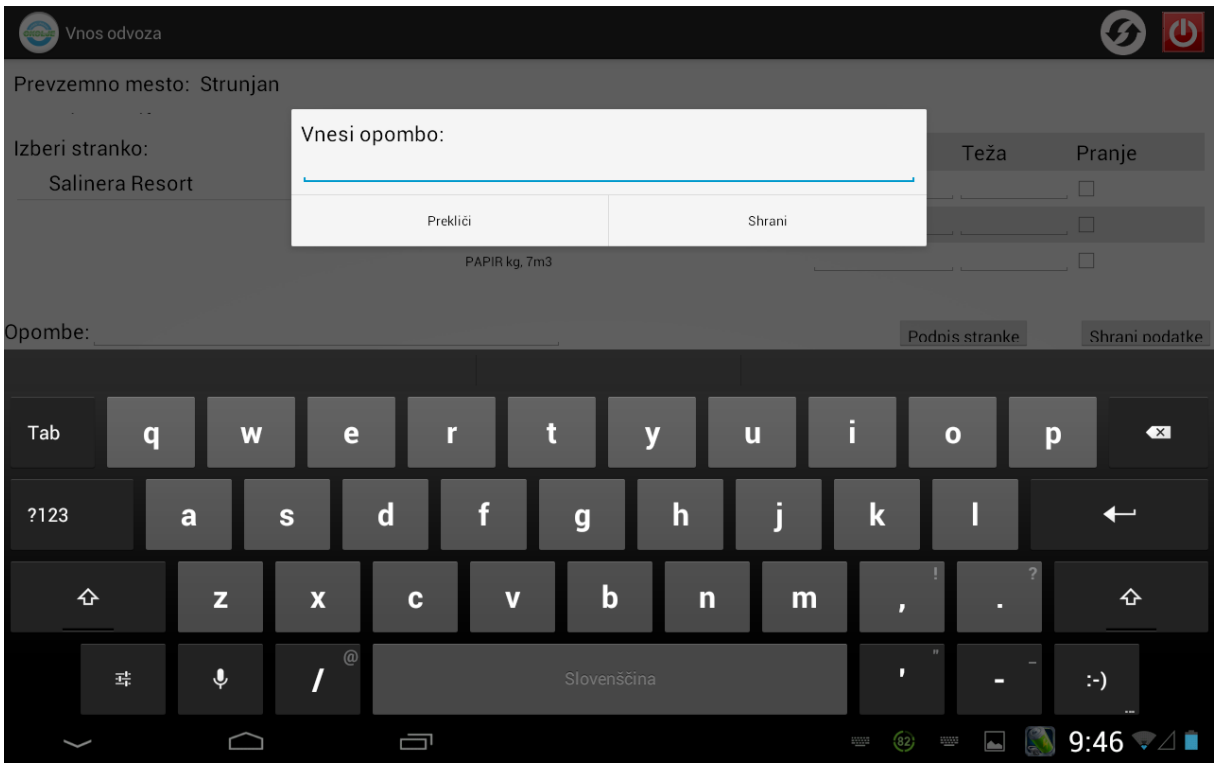
Koda 18: Primer dodajanja elementa v vrstico in nato v glavno tabelo.

V primeru klika na drugo stranko se tabela izbriše in nato ponovi postopek dinamičnega kreiranja vrstic z elementi. Prav tako se izbrišejo vsi trije seznami z elementi.

Na dnu aktivnosti (zelen okvir) je gumb za prehod v aktivnost za zajem podpisa in gumb za shranjevanje podatkov. Na levi imamo vnosno polje za morebitne opombe voznika pri določenem odvozu, ki pri kliku aktivira dialog. Dialog nam prikazuje gumb za preklic, gumb za shranjevanje opombe ter vnosno polje (Slika 4.12).
Ob pritisku na gumb *Shrani* se vneseni podatki shranijo v začasno spremenljivko, katero kasneje uporabimo pri shranjevanju v podatkovno zbirko.
S klikom na gumb *Podpis* preidemo v novo aktivnost, katero smo opisali v poglavju 5.7.4 AKTIVNOST PodpisActivity.
Pri povratku iz aktivnosti za zajem podpisa prikažemo informacijo o uspešnosti shranjevanja podpisa, v pojavnem oknu, katero smo pridobili iz splošnih nastavitev (Koda 19).

```
SharedPreferences sharedPrefs =
PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
String statusPodpisa= sharedPrefs.getString("status", null);
```

Koda 19: Pridobivanje informacij o uspešnosti shranjevanja podpisa iz splošnih nastavitev.



Slika 4.12: Opozorilni dialog, ki nam omogoča vnos opombe odvoza.

Gumb *Shrani podatke* sproži izvajanje največ funkcij v naši aplikaciji. Ob kliku nanj najprej shranimo vse vnesene količine in teže v tabelo nizov, pranja pa v tabelo podatkovnega tipa *boolean*. *S* pomočjo zanke to naredimo tolikokrat, kolikor je elementov v seznamih. Informacije o izbranem vozilu pridobimo iz splošnih nastavitev.

Če želimo shranjen podpis vnesti v odvoz, ga moramo najprej pridobiti iz začasne tabele, kar naredimo s stavkom *SQL* v metodi *getPodpis()* in ga shranimo v spremenljivko. Ko je enkrat shranjen v spremenljivki, ga lahko izbrišemo iz začasne tabele, saj je podpis individualen za vsak odvoz posebej in bo njegovo mesto zapolnil naslednji podpis.

Vnos opombe odvoza je izbiren in zato moramo preveriti, če je do vnosa prišlo. V primeru, da vnosa ni bilo, mu določimo vrednost praznega niza, v nasprotnem primeru pa imamo uporabnikov vnos. Ko imamo vse atribute za vnos odvoza pokličemo metodo *insertToSQLiteOdvoz(String izbranoVozilo, String ID_Stranke, String img_str, String opombe_odvoza)*, ki vnesene atribute shrani v tabelo *odvoz* (Koda 20).

```
.insertToSQLiteOdvoz(izbranoVozilo, ID_STRANKE, img_string, Opombe_Odvoza);

public void insertToSQLiteOdvoz(String izbranoVozilo, String ID_Stranke,
String img_str, String opombe_odvoza) {
	SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
	String currentDateandTime = sdf.format(new Date());
	SQLiteDatabase database = this.getWritableDatabase();
	String query =
	"INSERT INTO odvoz (Vozilo_IF_FK, Stranka_ID_FK, Datum_Cas,
	Podpis_Stranke, Opombe_Odvoza)
	VALUES ( ?, ?, ?, ?, ?)";
	String [] parameters = new String[5];
	parameters[0] = izbranoVozilo;
	parameters[1] = ID_Stranke;
	parameters[2] = currentDateandTime;
	parameters[3] = img_str;
	parameters[4] = opombe_odvoza;
	database.execSQL(query, parameters);
	database.close();
}
```

Koda 20: Klic s potrebnimi atributi za vstavljanje v tabelo *odvoz* in delovanje metode *insertToSQLiteOdvoz(String izbranoVozilo, String ID_Stranke, String img_str, String opombe_odvoza)*.

Med izvajanjem vstavljanja odvoza lovimo izjeme s pomočjo *try catch* bloka.
Če je odvoz uspešno vstavljen nadaljujemo z vstavljanjem v tabelo *odvoz_podrobno*. Uporabimo metodo *insertToSQLiteOdovzPodrobno* s katero vnesemo identifikacijsko številko odvoza, količino, težo in primarni ključ iz tabele s*tranka_posode*, ki služi kot tuji ključ v tabeli. Identifikacijsko številko odvoza pridobimo iz podatkovne zbirke na napravi. Iz rezultata metode *getPosode(String id_stranke)* izluščimo podatek o tujem ključu *Stranka_Posode_ID_FK* s pomočjo metode *pridobivanjeVECVrednostiIzListe.* Ta metoda kot argument sprejme seznam razpršenih tabel in ključ s katerim določimo katero vrednost želimo. Pridobi vse vrednosti (v tem primeru tujega ključa *Stranka_Posode_ID_FK*) in jih kot rezultat metode vrne v tabelo nizov (Koda 21).

```
public String[] pridobivanjeVECVrednostiIzListe( ArrayList<HashMap<String,
String>> arrayList, String key){
	String key1 = key;
	int index = 0;
	String[] arrayVrednosti = new String[arrayList.size()];
	for(HashMap<String, String> map : arrayList){
		Set<String> keys = map.keySet();
		for(String key2 : keys){
			if(key2 == key1){
				arrayVrednosti[index]= map.get(key);
			}
		}
		index++;
	}
	return arrayVrednosti;
}
```

Koda 21: Implementacija metode *pridobivanjeVECVrednostiIzListe.*

S pridobitvijo tujega ključa imamo vse potrebne argumente za vnos v tabelo *odvoz_podrobno.* V argumentih imamo tudi *List<EditText> vseKolicine,* ki služi za ugotavljanje števila obhodov *for* zanke. Količine, teže in pranja pošiljamo v tabelah iz katerih kasneje pridobimo posamezne vrednosti. Vstavljamo jih s pomočjo razreda *ContentValues,* ki shranjuje podatke v parih atribut-vrednost. V vsaki iteraciji *for* zanke pridobljene podatke shranimo v tabelo *odvoz_podrobno* kot nov zapis.

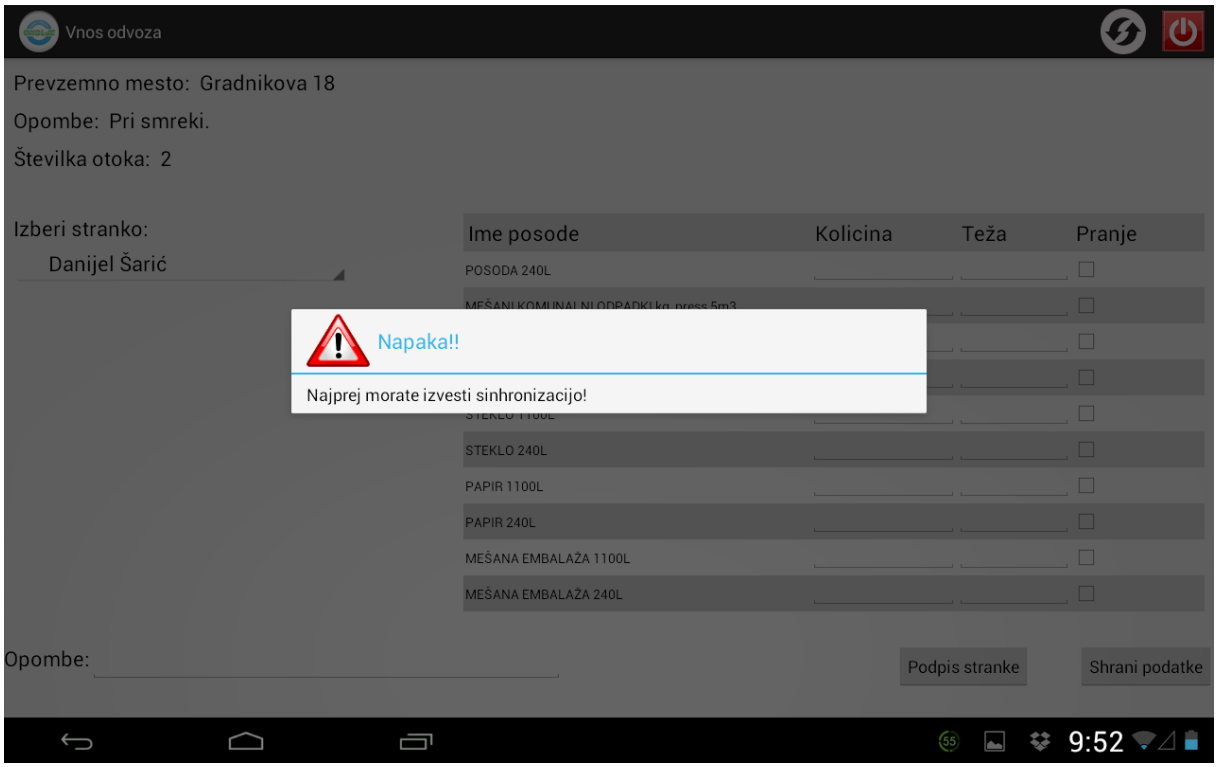
```
public void insertToSQLiteOdvozPodrobno(String IDOdvoza, List<EditText>
vseKolicine, String [] stringKolicine,  String [] stringTeze,   boolean []
vrednostCheckbox, String [] arrayIDStrankaPosode) {

	SQLiteDatabase database = this.getWritableDatabase();
	ContentValues values = new ContentValues();
	values.put("Odvoz_ID_FK", IDOdvoza);
	for(int i=0; i< vseKolicine.size(); i++){
		values.put("kolicina", stringKolicine[i]);
		values.put("teza",stringTeze[i]);
		values.put("pranje", vrednostCheckbox[i]+"");
		values.put("Stranka_Posode_ID_FK", arrayIDStrankaPosode[i]);
		database.insert("odvoz_podrobno", null, values);
	}
database.close();
}
```

Koda 22: Implementacija metode *insertToSQLiteOdvozPodrobno*.

Pri vstavljanju s pomočjo spremenljivk preverimo ali so bili podatki uspešno vstavljeni v obe tabeli in prikažemo ustrezno sporočilo uporabniku v pojavnem oknu *Toast*.

Ob uspešnem vnosu v obe tabeli se vsa vnosna polja počistijo za ponovno vnašanje. Uporabnik lahko ponavlja postopek tolikokrat, kolikor ima odvozov v tistem delovnem dnevu.

Po zaključenem delovnem dnevu in prihodu na sedež podjetja mora voznik opraviti sinhronizacijo podatkovnih zbirk in odjavo iz sistema v tem vrstnem redu. V primeru, da se želi uporabnik najprej odjaviti se mu prikaže opozorilni dialog, v katerem piše napaka (Slika 4.13).



Slika 4.13: Prikaz napake v opozorilnem diagramu pri poskusu odjave iz sistema pred sinhronizacijo podatkovnih zbrik.

O sinhronizaciji bomo povedali več v poglavju 5.9 SINHRONIZACIJA PODATKOVNE BAZE SQLITE Z PODATKOVNO BAZO MYSQL NA STREŽNIKU. V primeru uspešne sinhronizacije se uporabniku dovoli odjava. V metodi *onDestroy()* izbrišemo vse podatke iz splošnih nastavitev v izogib podvajanju podatkov pri naslednji prijavi.
Sistem zaključi z delovanjem in s pomočjo namere preide v aktivnost za prijavo.

4.7.4 AKTIVNOST PodpisActivity

PodpisActivity nam služi za zajemanje podpisa stranke. Implementacija zajema podpisa v novi aktivnosti se nam je zdela primerna, saj tako uporabniku omogočamo večjo površino za podpis.

Ko preidemo v aktivnosti za podpis se nam prikažejo trije gumbi: vnosno polje za ime in priimek ter platno (ang. Canvas) za podpis (Slika 4.14). Uporabnik mora najprej vnesti svoje ime in priimek, nato pa se lahko podpiše.

Slika 4.14: Prikaz grafičnega vmesnika aktivnosti *PodpisActivity*.

V primeru, da se pri podpisovanju zmoti, ima na voljo gumb *Izbriši,* ki počisti platno z metodo *clear()* iz podrazreda *signature()* v datoteki *PodpisActivity.java.* Lahko pride tudi do situacije, ko je potrebno preklicati zajem podpisa. V ta namen smo implementirali gumb *Prekliči,* ki se s pomočjo namere vrne nazaj na prejšnjo aktivnost (*OdvozActivity*) in podpisa ne shrani. Shrani pa status shranjevanja podpisa v splošne nastavitve. Ko je stranka zadovoljna s svojim podpisom ga lahko shrani s pritiskom na gumb *Shrani.* V tem primeru shrani tudi status shranjevanja podpisa v splošne nastavitve. Izvede se metoda *save(View v)* iz podrazreda *signature().* V tej metodi s pomočjo razreda *Bitmap* in njegove metode *createBitmpa(int width, int height, Bitmap.Config config)* shranimo vnesen podpis v spremenljivko s katero kasneje izvedemo pretvorbo zajetih podatkov v format *JPEG*. Za shranjevanje podpisa v lokalno zbirko smo uporabili *ByteArrayOutputStream* in kodirno shemo *Base64,* s katero smo pretvorili zajeti podpis v niz (ang. string) (Koda 23).

```
ByteArrayOutputStream stream = new ByteArrayOutputStream();
mBitmap.compress(Bitmap.CompressFormat.JPEG, 50, stream);
byte[] byte_arr = stream.toByteArray();
image_str = Base64.encodeToString(byte_arr, 0);
controller.insertToSQLitePodpis(image_str);
```

Koda 23: Shranjevanje podpisa v metodi *save(View v)*.

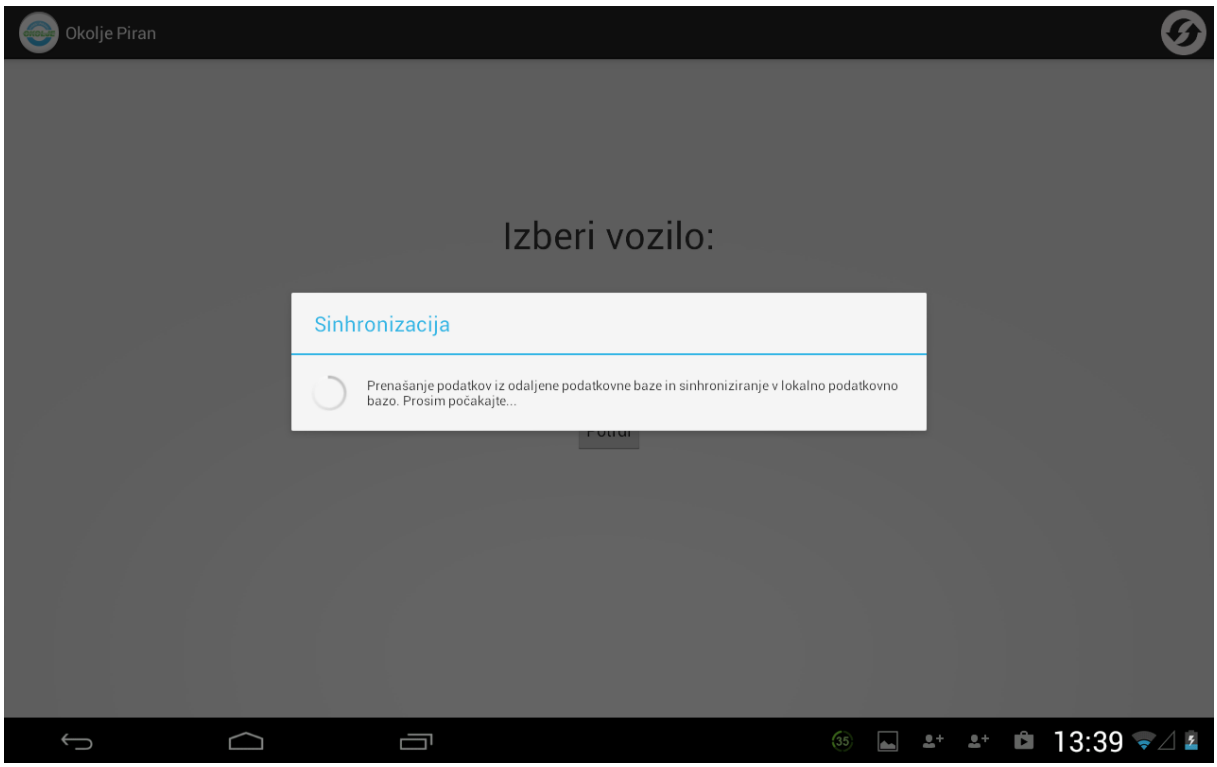
Po pretvorbi zajetega podpisa v niz smo rezultat shranili v trenutno tabelo *podpisi* v lokalni podatkovni zbirki z metodo *insertToSQLitePodpisString img_str()* (Koda 24).

```
public void insertToSQLitePodpis(String img_str) {
      SQLiteDatabase database = this.getWritableDatabase();
      String query = "INSERT INTO podpisi ( Podpis ) VALUES (?)";
      String [] parameters = new String[1];
      parameters[0] = img_str;
      database.execSQL(query, parameters);
      database.close();
}
```

Koda 24: Metoda za vstavljanje podpisov v razredu *DBHelper*.

4.8 SINHRONIZACIJA PODATKOVNE BAZE MYSQL Z PODATKOVNO BAZO SQLITE NA NAPRAVI

Z implementacijo sinhronizacije podatkovnih zbirk smo izboljšali delovanje podjetja. Omogočili smo hitrejše ažuriranje podatkov, saj se lahko zgodi, da pride do spremembe pri odvažanju odpadkov (npr. začetek sodelovanja z novo stranko). Če spremembo vnesemo pred voznikovo sinhronizacijo se izognemo nastanku nevšečnosti. Ob kliku na gumb za sinhronizacijo v aktivnosti za izbor vozila, se prične prenašanje podatkov iz oddaljene podatkovne zbirke na lokalno podatkovno zbirko. Kot rezultat mnogih operacij v ozadju se uporabniku prikaže pogovorno okno o napredku sinhronizacije, ki traja dokler se sinhronizacija ne zaključi (Slika 4.15).



Slika 4.15: Prikaz pogovornega okna o napredku sinhronizacije podatkovnih zbirk.

V asinhroni nalogi *Sinhronizacija* prikažemo pogovorno okno v metodi *onPreExecute()*. Ozadje asinhrone naloge se sklicuje na metode za sinhronizacijo vseh tabel iz strežnika na napravo. Predstavili bomo postopek sinhronizacije le tabele *stranka*, saj se vse ostale izvajajo po enakem postopku. V metodi *syncMySQLtoSQLiteDBStranka()* najprej kličemo datoteko *getStranka.php* z metodo *post(String url, RequestParams params, AsyncHttpResponseHandler responseHandler)* iz knjižnice *AsyncHttpClient* (Koda 25), ki nam vrne odgovor v formatu *JSON* (Koda 26).

```
AsyncHttpClient client = new AsyncHttpClient();
RequestParams params = new RequestParams();
client.post("http://danijelsaric.orgfree.com/okoljepiran/getStranka.php",
params, new AsyncHttpResponseHandler()
```

Koda 25: Prikaz sklicevanja na datoteko *getStranka.php*.

```
{
  "ID_Stranke": "10",
  "Ime_Stranke": "Danijel",
  "Priimek_Stranke": "Šarić",
  "Prevzemno_Mesto_Stranke": "Gradnikova 18",
  "St_Otoka": "2",
  "Opombe_Stranke": "Pri smreki."
}
```

Koda 26: Del odgovora datoteke *getStranka.php* v formatu JSON.

V primeru uspešnega odgovora izvedemo vstavljanje v podatkovno zbirko SQLite z metodo *insertStranka(JSONArray response)* iz razreda *DBHelper* (Koda 27).

```
String query = "INSERT INTO stranka ( ID_Stranke, Ime_Stranke,
Priimek_Stranke, Prevzemno_Mesto_Stranke,St_Otoka, Opombe_Stranke) " +
"VALUES (?,?,?,?,?,?)";
String [] parameters = new String[6];
parameters[0] = tmp.getString("ID_Stranke");
parameters[1] = tmp.getString("Ime_Stranke");
parameters[2] = tmp.getString("Priimek_Stranke");
parameters[3] = tmp.getString("Prevzemno_Mesto_Stranke");
parameters[4] = tmp.getString("St_Otoka");
parameters[5] = tmp.getString("Opombe_Stranke");
db.execSQL(query, parameters);
```

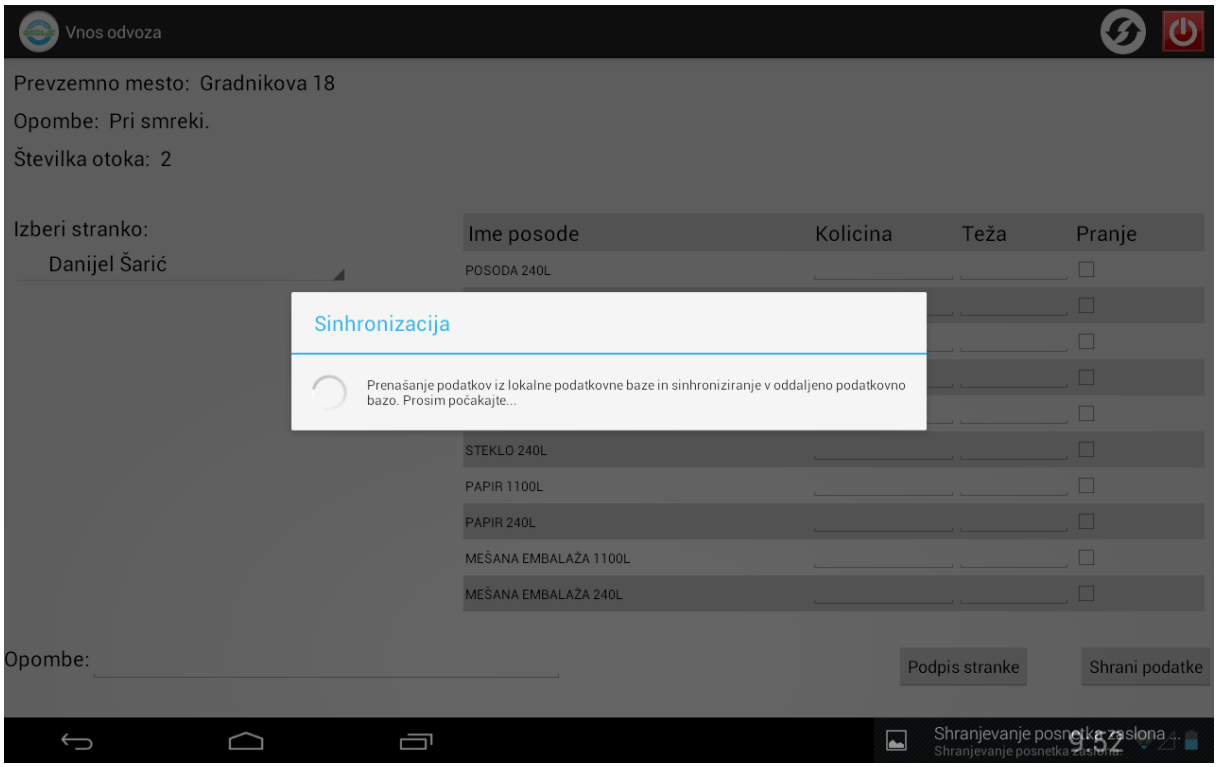
Koda 27: Stavek SQL in parametri s katerim vstavimo podatke v tabelo *stranka* na napravi.

Med izvajanjem sinhronizacije v spremenljivki shranjujemo informacijo o morebitni napaki, ki jo prikažemo uporabniku v pojavnem oknu *Toast* v metodi *onPostExecute(Void result)*. V primeru, da ni napak, zapremo pogovorno okno in preidemo v naslednjo aktivnost.

4.9 SINHRONIZACIJA PODATKOVNE BAZE SQLITE Z PODATKOVNO BAZO MYSQL NA STREŽNIKU

Voznik je med odvažanjem odpadkov shranjeval podatke v lokalno zbirko. Z našo implementacijo mobilne aplikacije izboljšamo ažurnost podatkovne zbirke, saj se posodablja dnevno in ne mesečno, kot do sedaj. S tem zmanjšamo čas in stroške shranjevanja podatkov v podatkovno zbirko na strežniku, saj ne potrebujemo človeškega faktorja za ročno vnašanje podatkov.

Tako kot pri prenašanju podatkov v obratni smeri tudi tukaj prikažemo pogovorno okno o napredku sinhronizacije podatkovnih zbirk (Slika 4.16).



Slika 4.16: Prikaz pogovornega okna o napredku pri sinhronizaciji podatkov iz lokalne podatkovne zbirke na oddaljeno.

Kllik na gumb za sinhronizacijo sproži izvajanje asinhrone naloge, ki v ozadju izvede le dve metodi in ne celotne zbirke kot v prejšnjem primeru. Ti metodi sta *syncSQLiteToMySQLDBOdvoz()* in *syncSQLiteToMySQLDBOdvozPodrobno()*.
Za uspešno vstavljanje vseh podatkov moramo najprej vstaviti tabelo *odvoz* in zatem še tabelo *odvoz_podrobno*, saj se tabeli navezujeta ena na drugo. Zato se bomo najprej osredotočili na delovanje metode za sinhroniziranje tabele *odvoz*.

Preveriti moramo ali obstajajo podatki v tabeli *odvoz*. V ta namen moramo uporabiti metodo *getAllOdvoz(),* ki vrne seznam razpršenih tabel z vsemi podatki od vseh odvozov v današnjem dnevu. Če podatkov ni, shranimo napako v spremenljivko. Uporabimo jo za prikaz informacije o uspešnosti sinhronizacije, v nasprotnem primeru pa nadaljujemo z izvajanjem. Za razliko od prejšnje sinhronizacije moramo sedaj pošiljati parametre, ki so podatki odvoza. Implementirali smo metodo *composeJSONfromSQLiteOdvoz(),* ki s pomočjo knjižnice *Gson* omogoči pretvorbo podatkov vseh dnevnih odvozov v format JSON.

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
String currentDateandTime = sdf.format(new Date());
ArrayList<HashMap<String, String>> odvoz_List;
odvoz_List = new ArrayList<HashMap<String, String>>();
String selectQuery =
"SELECT ID_Odvoz, Datum_Cas, Podpis_Stranke, Vozilo_IF_FK, Stranka_ID_FK,
Opombe_Odvoza
FROM odvoz
WHERE strftime('%Y-%m-%d', Datum_Cas)=strftime(?)";
//Parameter(?) je spremenljivka currentDateandTime
//Uporaba knjižnice Gson za pretvorbo podatkov v format JSON
Gson gson = new GsonBuilder().create();
return gson.toJson(odvoz_List);
```

Koda 28: Poizvedba SQL za prikaz odvozov današnjega dne in uporaba knjižnice *Gson*.

S pretvorjenimi parametri se povežemo na strežnik in sicer na datoteko *insertOdvozMySQL.php*. Vstavljanje v podatkovno zbirko na strežniku izvedemo v *for* zanki z *insert into* stavkom SQL ter poslanimi parametri (Koda 29). Kot odgovor vrnemo informacijo o uspešnosti vstavljanja. Te informacije na napravi prikažemo uporabniku s pomočjo spremenljivk in pojavnega okna *Toast*.

```
$query = "insert into odvoz
( Datum_Cas,Podpis_Stranke,Vozilo_IF_FK,Stranka_ID_FK,Opombe_Odvoza )
Values
( :Datum_Cas,:Podpis_Stranke,:Vozilo_IF_FK,:Stranka_ID_FK, :Opombe_Odvoza )";

$query_params = array(
':Datum_Cas' => $data[$i]->Datum_Cas,
':Podpis_Stranke' => $data[$i]->Podpis_Stranke,
':Vozilo_IF_FK' => $data[$i]->Vozilo_IF_FK,
':Stranka_ID_FK' => $data[$i]->Stranka_ID_FK,
':Opombe_Odvoza' => $data[$i]->Opombe_Odvoza );
```

Koda 29: Prikaz načina vstavljanja podatkov v podatkovno zbirko na strežniku.

Po uspešno vnesenih podatkih iz tabele *odvoz* lahko nadaljujemo z vstavljanjem podatkov iz tabele *odvoz_podrobno*.

Ravno tako kot v prejšnji, tudi v metodi *syncSQLiteToMySQLDBOdvozPodrobno()* preverimo, če obstajajo podatki v tabeli. Uporabimo metodo *getAllOdvozPodrobno()*, ki vrne vse vnešene podatke iz tabele *odvoz_podrobno*.

Pri prenosu podatkov iz oddaljene podatkovne zbirke na lokalno, lahko več voznikov hkrati prenese iste podatke, kar bi lahko pripeljalo do kasnejših težav pri vstavljanju podatkov v podatkovno zbirko na strežniku. Tabela *odvoz_podrobno* vsebuje tuji ključ *Odvoz_ID_FK*. Ko prenašamo podatke na napravo prenesemo pretekle odvoze in s tem dobimo tudi podatek o zadnjem identifikacijskem številu odvoza, ki se avtomatično povečuje. Problem nastane takrat, kadar želimo vstaviti nov odvoz v zbirko na strežniku, saj imajo vsi vozniki preneseno enako številko zadnjega odvoza. Vozniki bi poskušali vstaviti odvoz v naslednjo številko zadnjega odvoza, kar bi pripeljalo do težav, saj ni mogoče vstaviti več odvozov pod eno številko.
V izogib težavam pri vstavljanju tabele *odvoz* v podatkovno zbirko na strežniku enostavno izpustimo argument *ID_Odvoz*, saj se avtomatsko zgenerira. Vendar pa pri tabeli *odvoz_podrobno* nimamo pravilne vrednosti. Tuj ključ *Odvoz_ID_FK*, prek katerega vršimo povezavo med tabelama, bi lahko bil napačen, kar bi rezultiralo povezavo na napačen odvoz.

Zato v tem primeru pošiljamo dva parametra in sicer podatke iz tabele *odvoz* in tabele *odvoz_podrobno*. Povezavo s strežnikom izvršimo z datoteko *insertOdvoz_podrobnoMySQL.php,* v kateri uporabimo poslane parametre. Najprej sprejmemo parametre o odvozu. S *select* stavkom SQL v *for* zanki pridobimo identifikacijsko število vnesenega odvoza pri točno določeni stranki, s točno določenim vozilom v določenem času, katero shranimo v spremenljivko (Koda 30). Na ta način onemogočimo vstavljanje v tabelo *odvoz_podrobno* z napačnim tujim ključem. To pa zato, ker je nemogoče, da imamo dva odvoza z identičnimi podatki.

```
$query =
"select ID_Odvoz from odvoz
 where Datum_Cas= :Datum_Cas
 and Vozilo_IF_FK = :Vozilo_IF_FK and Stranka_ID_FK = :Stranka_ID_FK";

$query_params = array( ':Datum_Cas' => $data_odvoz[$J]->Datum_Cas,
':Vozilo_IF_FK' => $data_odvoz[$J]->Vozilo_IF_FK,
':Stranka_ID_FK' => $data_odvoz[$J]->Stranka_ID_FK);
```

Koda 30: Poizvedba s katero pridobimo vrednost identifikacijskega števila.

Znotraj prve *for* zanke naredimo še eno *for* zanko v kateri sprejmemo parametre tabele *odvoz_podrobno*. Ker imamo zanko v zanki se lahko podvojijo vnosi v tabelo *odvoz_podrobno*. Zato smo na začetku druge zanke preverjali enakost med poslanima parametrom *ID_Odvoz* iz tabele *odvoz* in parametrom *Odvoz_ID_FK* iz tabele *odvoz_podrobno*. V primeru neenakosti smo preskočili eno iteracijo zanke, v nasprotnem primeru pa smo vstavili podatke v tabelo *odvoz_podrobno* s tujim ključem iz spremenljivke.

```
if($data[$i]->Odvoz_ID_FK != $data_odvoz[$J]->ID_Odvoz)
CONTINUE;
$query = "insert into odvoz_podrobno
(Kolicina, Teza, Odvoz_ID_FK, Stranka_Posode_ID_FK, Pranje)
values (:Kolicina,:Teza,:Odvoz_ID_FK,:Stranka_Posode_ID_FK,:Pranje)";

//$Odvoz_ID je spemenljivka v katero smo shranili rezulat prejšne poizvedbo
$query_params = array( ':Kolicina' => $data[$i]->Kolicina,
':Teza' => $data[$i]->Teza, ':Odvoz_ID_FK' => $Odvoz_ID,
':Stranka_Posode_ID_FK' => $data[$i]->Stranka_Posode_ID_FK,
':Pranje' => $data[$i]->Pranje);
```

Koda 31: Preverjanje enakosti in vstavljanje podatkov v tabelo *odvoz_podrobno*.

Kot pri vstavljanju v tabelo *odvoz* tudi tukaj po vstavljanju v tabelo *odvoz_podrobno* v podatkovni zbirki na strežniku prikažemo uporabniku informacijo o uspešnosti sinhronizacije podatkov.

4.10 TESTIRANJE

Pri razvijanju aplikacije je zelo pomembno testiranje, saj želimo le-tej dodati vrednost. Dodajanje vrednosti pomeni višanje kakovosti in zanesljivosti aplikacije. Ne želimo si nikakršnih napak pred končno verzijo aplikacije, saj bi lahko za kasnejše odpravljanje napak porabili veliko več časa in denarja.

Aplikacijo smo testirali med samim razvojem, kar pomeni predvsem testiranje napisanih funkcij v programski kodi. Želeli smo se prepričati, da na novo dodane funkcije delujejo pravilno in opravljajo pričakovano delo. Na ta način nismo odkrili vseh napak ampak smo njihovo število zmanjšali. Testiranje smo izvajali sami na napravi Samsung Tab 2 10.1 z operacijskim sistemom Android verzije 4.1.2, saj smo ugotovili, da z emulatorjem porabimo preveč časa za zagon testiranja. Tablični računalnik smo povezali z osebnim računalnikom in na tablici omogočili odpravljanje težav s povezavo USB. Potek izvajanja aplikacije smo opazovali v Log zavihku razvojnega okolja Eclipse.

5 SKLEPNE UGOTOVITVE

V okviru diplomskega dela smo razvili mobilno aplikacijo za tablične računalnike z operacijskim sistemom Android in podatkovno zbirko na strežniku, ki shranjuje zajete podatke v mobilni aplikaciji. Cilj diplomskega dela je bil razviti Android aplikacijo, ki bi poenostavila ter zmanjšala delo voznikom tovornjakov komunalnih odpadkov. Prototip mobilne aplikacije realizira zahteve podjetja. V diplomskem delu smo predstavili delovanje sistema od ideje do končnega delujočega paketa aplikacij. Predstavili smo tudi pomembnejše dele aplikacije z izvorno kodo, ki poskrbi za delovanje aplikacije. Opisali smo večino metod s pomočjo katerih je bilo omogočeno izvajanje določenih fukncionalnosti.

Največ težav smo imeli pri impelementaciji sinhronizacije podatkovnih zbirk. Nekaj težav nam je na začetku razvoja povzročala tudi komunikacija med odjemalcem in strežnikom.

Ugotovili smo, da je uradna dokumentacija operacijskega sistema Android dobro podprta. Zaradi dobre podprtosti obstaja veliko število razvijalcev in s tem tudi veliko število primerov s katerimi smo pridobili znanje za razvijanje aplikacij. Zaradi neizkušenosti smo pri implementaciji nekaterih navidez enostavnih rešitev porabili veliko več časa kot pričakovano.

Za optimalno delovanje aplikacije je potrebno vložiti še nekaj dela, saj imamo še veliko prostora za izboljšavo. Lahko bi izboljšali izgled in funkcionalnost grafičnega vmesnika ter varnost aplikacije in podatkovne zbirke na strežniku.

Pri izvajanju diplomskega dela smo pridobili veliko novih izkušenj ter ogromno novega znanja, kar nam bo prišlo prav v prihodnosti. Veliko smo se naučili o delovanju operacijskega sistema Android, pa tudi o delovanju spletnih storitev.

LITERATURA

[1] (2015) Java.
Dostopno na: http://en.wikipedia.org/wiki/Java_%28programming_language%29.

[2] (2015) MySQL.
Dostopno na: http://en.wikipedia.org/wiki/MySQL.

[3] (2015) SQL.
Dostopno na: http://en.wikipedia.org/wiki/SQL.

[4] (2015) SQLite.
Dostopno na: http://en.wikipedia.org/wiki/SQLite.

[5] (2015) XML.
Dostopno na: http://en.wikipedia.org/wiki/XML.

[6] (2015) PHP.
Dostopno na: http://sl.wikipedia.org/wiki/PHP.

[7] (2015) REST.
Dostopno na: http://en.wikipedia.org/wiki/Representational_state_transfer.

[8] (2015) JSON.
Dostopno na: http://json.org/json-sl.html.

[9] (2015) Android operating system.
Dostopno na: http://en.wikipedia.org/wiki/Android_%28operating_system%29.

[10] (2015) Global market share held by smartphone operating system from 2009 to 2014.
Dostopno na: http://www.statista.com/statistics/263453/global-market-share-held-by-smartphone-operating-systems/.

[11] (2015) Number of apps available in leading app stores as of July 2014.
Dostopno na: http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

[12] (2015) Eclipse software.
Dostopno na: http://en.wikipedia.org/wiki/Eclipse_%28software%29.

[13] (2015) Android software development.
Dostopno na: http://en.wikipedia.org/wiki/Android_software_development.

[14] (2015) Eclipse ADT.
Dostopno na: http://developer.android.com/tools/sdk/eclipse-adt.html.

[15] (2015) DBDesigner.
Dostopno na: http://dbdesigner.sourceforge.net/.

[16] (2015) PHPMyAdmin.
Dostopno na: http://www.phpmyadmin.net/home_page/index.php.

[17] (2015) AsyncTask.
Dostopno na: http://programmerguru.com/android-tutorial/what-is-asynctask-in-android/.

[18] (2015) Graphical user interface.
Dostopno na: http://en.wikipedia.org/wiki/Graphical_user_interface.

[19] (2015) Android activities.
Dostopno na: http://developer.android.com/guide/components/activities.html.

[20] L. Dacey, S. Conder. Sams Teach Yourself Android Application Development in 24 hours, Sams, United States of America, 2010.

SLIKE

Slika 4.1: Arhitektura sistema. 15
Slika 4.2: Prikaz asinhronega delovanja. 17
Slika 4.3: Entitetno relacijski model podatkovne zbirke. 18
Slika 4.4: Primer poizvedbe tabele *odvoz_podrobno*. 20
Slika 4.5: Življenski cikel aktivnosti in njihovih stanj. 23
Slika 4.6: Diagram aktivnosti. 24
Slika 4.7: Prijava voznika. 25
Slika 4.8: Aktivnost VozilaActivity ob zagonu. 27
Slika 4.9: Prikaz podatkov v izvlečnem seznamu. 28
Slika 4.10: Opozorilni dialog z napisom vrste napake. 28
Slika 4.11: Glavna aktivnost *OdvozActivity*. 29
Slika 4.12: Opozorilni dialog, ki nam omogoča vnos opombe odvoza. 33
Slika 4.13: Prikaz napake v opozorilnem diagramu pri poskusu odjave iz sistema pred sinhronizacijo podatkovnih zbrik. 36
Slika 4.14: Prikaz grafičnega vmesnika aktivnosti *PodpisActivity*. 37
Slika 4.15: Prikaz pogovornega okna o napredku sinhronizacije podatkovnih zbirk. 39
Slika 4.16: Prikaz pogovornega okna o napredku pri sinhronizaciji podatkov iz lokalne podatkovne zbirke na oddaljeno. 41